Machine Learning for Preconditioning Elliptic Equations in Porous Microstructures: A Path to Error Control

Kangan Li, Sabit Mahmood Khan, Yashar Mehmani

Energy and Mineral Engineering Department, The Pennsylvania State University, University Park, Pennsylvania 16802

Abstract

Elliptic equations on complex porous microstructures govern the flow of fluids inside subsurface rocks in underground CO_2 and H_2 storage, and the transport of heat and solute within electrochemical devices like batteries and fuel cells. The algebraic systems arising from the discretization of these equations are often prohibitively large and must be solved via iterative (e.g., Krylov) methods, for which effective preconditioning is key to ensure rapid convergence. In recent work, the authors proposed a scalable two-level preconditioner whose performance was superior to existing algebraic multigrid variants for pore-scale problems. The preconditioner was based on the pore-level multiscale method (PLMM) and consisted of a coarse preconditioner, $\mathrm{M_G}$, and a fine smoother, $\mathrm{M_L}$. Similar two-level preconditioners based on the multiscale finite element/volume and variational multiscale methods also exist for solving continuum-scale PDEs in porous media. The most expensive step in building such two-level preconditioners is computing $\mathrm{M_G}$, for which many numerical bases on a set of subdomains must be calculated to yield a prolongation matrix. Here, we show that machine learning (ML) can dramatically reduce this cost. Moreover, by embedding ML within a preconditioning framework, we enable the rarity of estimating and controlling ML errors to any desired level. We systematically probe the ML-built preconditioner in solving the Poisson and linear-elasticity equations over complex 2D/3D geometries and show that it performs comparably to its solver-built counterpart. Implications and future extensions are discussed.

Keywords: Porous media, Pore scale, Multiscale method, Machine learning, Preconditioning, Elliptic equations

1. Introduction

In the context of porous media physics, elliptic equations describe the flow of fluids, transport of heat and solute through the intricate void space, and mechanical deformation of the solid matrix. Understanding and controlling these processes is important to the high-precision engineering of geologic CO_2 sequestration [1], underground H_2 storage [2], geothermal energy extraction [3], and the optimal design and operation of fuel cells [4] and electrolyzers [5] for energy storage and conversion. Prior to solving such PDEs, the microscale geometry of a porous sample is mapped experimentally via, e.g., an X-ray μCT scanner [6, 7]. The acquired image is then passed as input to a *pore-scale model* that discretizes and solves the PDE. The highest fidelity among pore-scale models are direct numerical simulation (DNS) techniques, e.g., the finite element (FEM), finite volume (FVM), and finite difference (FDM) methods [8]. Here, we focus on scalar- and vector-valued PDEs relevant to heat conduction and elastic deformation, respectively.

Given the need to analyze statistically representative, thus large, samples, the size of the linear(ized) systems obtained from discretizing the above PDEs is often enormous. This, in turn, demands iterative (e.g., Krylov) methods for solving such systems, whose rapid convergence hinges upon the availability of effective preconditioners [9]. One very successful preconditioner is the algebraic multigrid (AMG) method [10], and its many variants (e.g., [11]). Viewed as a solver, AMG operates by reducing the original system, $\mathrm{A}x = b$, to a smaller coarse system, $\mathrm{A}^c x^c = b^c$, that is faster to solve. The coarse solution, x^c, is then interpolated (or prolongated) onto the original fine grid to yield an approximate solution, $\tilde{x}$. Errors in $\tilde{x}$ are dominated by high-frequency modes, which are further attenuated with a

*Corresponding author: Yashar Mehmani. Email: yzm5192@psu.edu
Email addresses: kbl5610@psu.edu (Kangan Li), skk6071@psu.edu (Sabit Mahmood Khan), yzm5192@psu.edu (Yashar Mehmani)

smoother (e.g., ILU(k), Gauss-Seidel). The foregoing is a *two-level* outline of AMG that consists of a *fine* and a *coarse* system/grid. Incorporating additional levels is straightforward by repeating these steps in nested fashion. Viewed as a preconditioner for accelerating Krylov solvers, AMG consists of two parts: (1) a coarse preconditioner, $\mathrm{M_G}$, used to build and solve the coarse system, and (2) a fine smoother, $\mathrm{M_L}$, used to attenuate high-frequency errors. While both are important, AMG's success depends critically on the quality of the coarsened matrix $\mathrm{A^c}$ and vector b^c.

The coarsening requires a prolongation, P, and a restriction, R, matrix that allow accurate mapping to/from the fine and coarse vector spaces. These matrices are used to yield $\mathrm{A^c} = \mathrm{P}\,(\mathrm{RAP})^{-1}\mathrm{R}$ and $b^c = \mathrm{R}\,b$ [9]. Computing R is often cheap, since it is either taken as the transpose of P, if A is symmetric as in FEM [10], or built trivially out of 0 and 1 entries to enable certain row-sum operations, if A originates from FVM [12]. The critical and costly step corresponds to building P. An accurate P consists of columns, called *bases*, whose span contains a very close approximation to x. Given AMG is a black-box preconditioner, its P is not always optimal, leading to subpar performance [13]. This has spurred the development of more physics/geometry-informed preconditioners [14–18] based on two-level solvers such as multiscale finite element (MsFE) [19, 20], multiscale finite volume (MsFV) [21, 22], and mixed mortar finite element [23, 24]. The columns of P in such methods are obtained by solving local problems on small subdomains, subject to carefully crafted closure boundary conditions (BCs). Despite their superior performance, these multiscale preconditioners are designed for PDEs describing continuum-scale (or Darcy) physics of porous media.

Recently, the authors have proposed a two-level preconditioner [25, 26] based on the pore-level multiscale method (PLMM) [27, 28] for elliptic equations arising from linear-elastic mechanics at the pore scale. A similar preconditioner was later formulated for saddle-point systems associated with the Stokes flow equation [29]. The preconditioners were shown to exhibit far superior performance to AMG in solving pore-scale problems. PLMM itself is a two-level solver that was first developed for single-phase flow [30] and later extended to two-phase flow [31], compressible flow [32], and elastic deformation of intact/fractured porous media [27, 28, 33]. It consists of four main steps: (1) decompose the domain into subdomains by cutting it at geometric constrictions using the watershed segmentation algorithm [34]; (2) compute local basis functions on each subdomain subject to closure BCs; (3) solve a coarse problem that imposes flux continuity across all subdomain interfaces; and (4) iterate to improve the accuracy of the closure BCs, and thereby that of the solution. The preconditioner by [27, 28], referred to hereafter as M, interprets the above steps in a purely algebraic fashion, and much like AMG, consists of a coarse, $\mathrm{M_G}$, and fine, $\mathrm{M_L}$, preconditioner. Computing $\mathrm{M_G}$, or equivalently its prolongation matrix P, is the most expensive step in building M as it requires calculating multiple basis functions per subdomain. A similar upfront cost is incurred by *all* two-level preconditioners discussed above.

Our goal is to accelerate the construction of $\mathrm{M_G}$ in the PLMM preconditioner, M, via supervised machine learning (ML). Specifically, we propose a convolutional encoder-decoder neural network, based on the U-Net [35] and ResNet [36] architectures, that accepts the geometry of a subdomain as input (in the form of a small image) and yields the basis functions defined on it as output. The bases are then assembled into the columns of the prolongation matrix P for $\mathrm{M_G}$. While our quest may seem esoteric and specific to PLMM, it is not, and has wide ranging implications. Existing literature on ML for solving PDEs in porous media [37] is encumbered by two fundamental drawbacks: (1) Effective mechanisms for estimating and controlling prediction errors are lacking [38]. ML outputs must either be accepted at face value, or trusted based on empiricism or criteria that determine whether a sample is in- or out-of-distribution; (2) Training is performed on *whole* (non-decomposed) domains, often in the form of large, geometrically complex 2D/3D images [39–43]. Compared to the roughly convex subdomains obtained from watershed segmentation herein, training on whole domains is less desirable because: (a) the statistical space of whole-domain microstructures is much larger, thus demanding more training data; (b) labeled data for whole domains are scarcer, requiring large X-ray images and costly DNS simulations. By contrast, one whole domain yields hundreds of subdomains (i.e., data) when decomposed, that can be rapidly processed via DNS; (c) ML algorithms for whole domains do not generalize well to BCs beyond which they are trained, while basis functions can be assembled in any combination to enforce arbitrary BCs.

In this work, we address these drawbacks. By tasking our ML algorithm to build a coarse preconditioner, $\mathrm{M_G}$, and pairing it with a smoother, *we enable error estimation and control.* Moreover, by training on semi-convex subdomains, we ensure the algorithm is less data hungry, that data generation is easier, and arbitrary BCs can be flexibly imposed. The ML algorithm's outputs are also in a sense *reusable*, because basis functions built for linear PDEs, such as small-strain elasticity, can be used to solve (linearized forms of) nonlinear PDEs, such as finite-strain deformation and plasticity [44]. We systematically probe the ML algorithm in building coarse preconditioners for the Poisson and linear-elasticity equations defined on complex 2D/3D porous geometries. Compared to purely solver-built preconditioners, the cost of building $\mathrm{M_G}$ is dramatically reduced while the convergence rate of the Krylov solver is minimally degraded. We also demonstrate that if trained on simple disk/sphere packs, the ML algorithm is transferable to more complex geometries (e.g., sandstone, bone) without additional training.

In closing, we remark that two other attempts [45, 46] at using ML to accelerate the construction of basis functions for elliptic equations have been made: one in the context of MsFV, and another in the context of an MsFV variant called MsRSB [47]. However, given their focus lies in uncertainty quantification at the Darcy scale, both treat scalar-valued PDEs (i.e., the pressure equation) on subdomains that are square shaped. Neither address error control, vector-valued equations, or how to handle arbitrary pore-scale geometries, all specific aims of the present work.

The paper is organized as follows: Section 2 describes the PDEs to be solved on porous microstructures. Section 3 briefly reviews the PLMM preconditioner consisting of $\mathrm{M_G}$ and $\mathrm{M_L}$. We discuss the ML architecture for building $\mathrm{M_G}$ in Section 4. Sections 5 and 6 present a series of 2D/3D numerical tests to probe the ML-enhanced preconditioner. In Section 7, we discuss the implications of the results and future directions. Section 8 concludes the paper.

2. Problem description

We target two PDEs with our ML-assisted preconditioning described later: (1) scalar-valued Poisson equation, and (2) vector-valued linear-elasticity equation. Consider a porous domain $\Omega \subset \mathbb{R}^D$ with Lipschitz boundary $\partial\Omega$, where D is the number of spatial dimensions. Such a domain is represented here by a pore-scale image (e.g., X-ray μCT) as shown in Fig.1a (gray means solid). The boundary $\partial\Omega$ consists of the void-solid interface, Γ^w, and the external surface (or bounding box) of the domain, Γ^{ex}. Another way to partition $\partial\Omega$ is into Dirichlet, Γ^d, and Neumann, Γ^n, segments such that $\partial\Omega = \Gamma^d \cup \Gamma^n$ and $\Gamma^d \cap \Gamma^n = \emptyset$ hold. We assume $\Gamma^w \subset \Gamma^n$, implying a stress-/flux-free fluid-solid interface.

The Poisson equation reads as follows:

$$-\Delta u = f\ , \qquad \text{on } \Omega \tag{1a}$$

$$u = u_d\ , \qquad \text{on } \Gamma^d \tag{1b}$$

$$\nabla u \cdot \boldsymbol{n} = t_n\ , \qquad \text{on } \Gamma^n \tag{1c}$$

where we seek the scalar solution u, subject to the known source term f, the prescribed function value u_d on Γ^d, and the prescribed flux t_n on Γ^n. The vector $\boldsymbol{n}$ denotes the outward-pointing unit normal on Γ^n.

The linear-elasticity equation is given by:

$$-\nabla \cdot \boldsymbol{\sigma}(\boldsymbol{u}) = \boldsymbol{f}\ , \qquad \text{on } \Omega \tag{2a}$$

$$\boldsymbol{u} = \boldsymbol{u}_d\ , \qquad \text{on } \Gamma^d \tag{2b}$$

$$\boldsymbol{\sigma}(\boldsymbol{u})\boldsymbol{n} = \boldsymbol{t}_n\ , \qquad \text{on } \Gamma^n \tag{2c}$$

where we seek the vector solution $\boldsymbol{u}$, subject to the body force $\boldsymbol{f}$, the prescribed displacement $\boldsymbol{u}_d$ on Γ^d, and the prescribed traction $\boldsymbol{t}_n$ on Γ^n. The Cauchy stress tensor $\boldsymbol{\sigma}$ is related to the displacement field $\boldsymbol{u}$ via:

$$\boldsymbol{\sigma}(\boldsymbol{u}) = \mathbf{C} : \boldsymbol{\varepsilon}(\boldsymbol{u}) \tag{3}$$

where $\boldsymbol{\varepsilon}(\boldsymbol{u}) = \nabla^s \boldsymbol{u} = 1/2\,(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^\top)$ is the strain tensor and $\mathbf{C} = [\mathsf{C}_{ijkl}]$ the fourth-order stiffness tensor. The symbol ∇^s denotes the symmetric gradient operator and the superscript $\top$ denotes transposition. For an isotropic material, $\mathbf{C}$ is:

$$\mathsf{C}_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) \tag{4}$$

where λ and μ are Lamé parameters. Substituting Eq.4 into Eq.3 yields:

$$\boldsymbol{\sigma}(\boldsymbol{u}) = \lambda\,\mathrm{tr}(\boldsymbol{\varepsilon}(\boldsymbol{u}))\boldsymbol{I} + 2\mu\,\boldsymbol{\varepsilon}(\boldsymbol{u}) \tag{5}$$

where $\mathrm{tr}(\boldsymbol{\varepsilon})$ represents the trace of $\boldsymbol{\varepsilon}$.

In this work, Eqs.1 and 2 are discretized with a Galerkin finite element method (FEM) over a Cartesian mesh that conforms to, or is an integer fraction of, the image pixels comprising Ω. In other words, elements are rectangu-

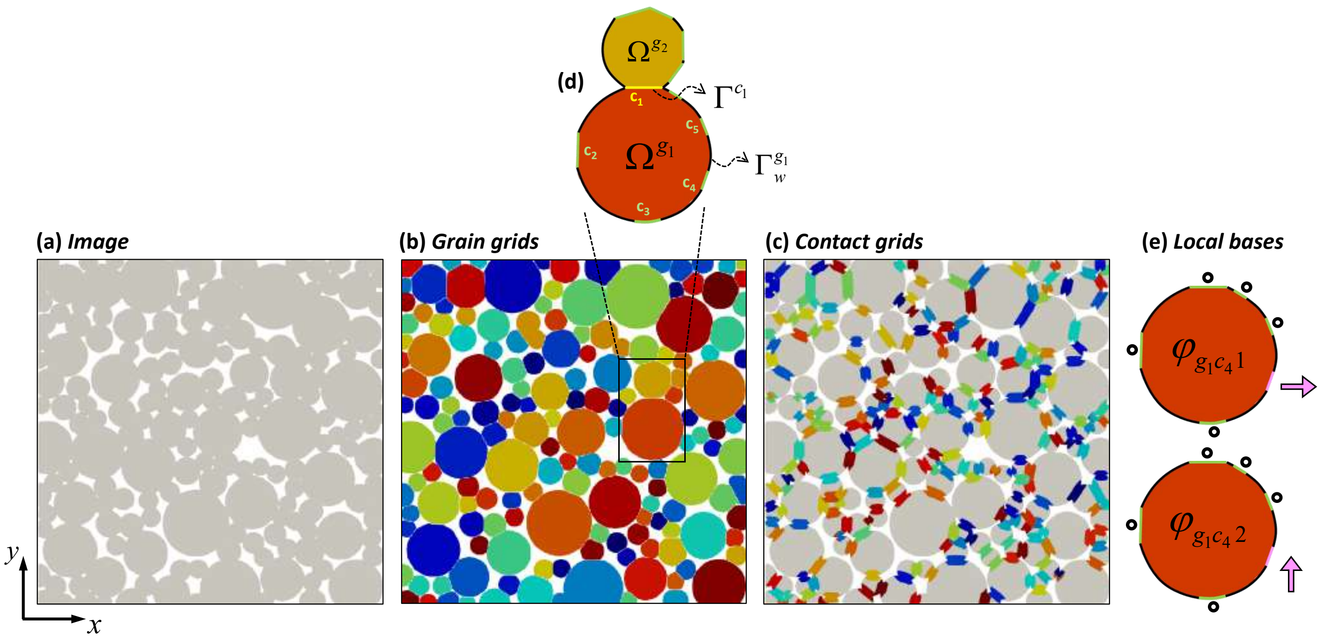

Figure 1: Schematic of a pore-scale image, its decomposition into grain grids, and contact grids. (a) The image consists of a solid phase Ω (gray) and a void space (white). The solid is where the PDEs in Eqs.1 and 2 are solved. (b) Ω is decomposed into grain grids Ω^{g_i} (randomly colored). (c) Contact grids Ω^{ζ_k} cover a thin region around each contact interface, Γ^{c_j}, shared between adjacent grain grids. (d) An interface Γ^{c_1} (yellow) between two grain grids Ω^{g_1} and Ω^{g_2} is highlighted. (e) Two basis functions associated with Ω^{g_1} and Γ^{c_4} are shown for the linear elasticity PDE. Black circles denote homogeneous Dirichlet BCs, and arrows are Dirichlet BCs of unit magnitude along the coordinate axes.

lar/cuboid and FEM shape functions are bilinear/trilinear in 2D/3D. This yields the following linear system:

$$\hat{\mathrm{A}}\hat{x} = \hat{b} \tag{6}$$

where $\hat{\mathrm{A}}$ is the coefficient matrix, $\hat{b}$ is the right-hand side (RHS) vector, and $\hat{x}$ the unknown vector of FEM nodal values corresponding to u in Eq.1 or $\boldsymbol{u}$ in Eq.2. Solving Eq.6 on typical domain sizes of interest is computationally expensive, especially when Ω is large and geometrically complex. This can even lead state-of-the-art algebraic multigrid (AMG) [10] solvers to converge slowly [25, 26]. Below, we first review the highly effective PLMM preconditioner, M, for such problems that accelerates the convergence of Krylov solvers. We then describe how machine learning (ML) can be used to accelerate a key computational bottleneck in building M (i.e., $\mathrm{M_G}$). Neither M nor its hybridization with ML are limited to FEM or Cartesian grids, as they apply to other solvers (e.g., FVM) and unstructured grids.

3. Multiscale preconditioner based on PLMM

We briefly review the multiscale preconditioner, M, based on PLMM [26]. Section 3.1 outlines the overall structure of M, consisting of a global (or coarse) preconditioner, $\mathrm{M_G}$, and a smoother, $\mathrm{M_L}$. We then describe the domain decomposition that is central to constructing both $\mathrm{M_G}$ and $\mathrm{M_L}$. Section 3.3 details the various building blocks of $\mathrm{M_G}$, which is the target of our acceleration via machine learning. In Section 3.4, we only summarize a few essential points about $\mathrm{M_L}$, as the details are not central to this work. We supplement this with references for the interested reader.

3.1. Overall structure

The PLMM preconditioner, M, is formulated as follows:

$$\mathrm{M}^{-1} = \mathrm{M_G}^{-1} + \mathrm{M_L}^{-1}(\mathrm{I} - \hat{\mathrm{A}}\mathrm{M_G}^{-1}) \tag{7}$$

where the global preconditioner $\mathrm{M_G}$ attenuates low-frequency errors, and the smoother $\mathrm{M_L}$ removes high-frequency errors. Eq.7 is a multiplicative combination, where $\mathrm{M_G}$ is applied first, and $\mathrm{M_L}$ next. The smoother itself is written as:

$$\mathrm{M_L}^{-1} = \sum_{i=1}^{n_{st}} \mathrm{M}_l^{-1} \prod_{j=1}^{i-1} (\mathrm{I} - \hat{\mathrm{A}}\mathrm{M}_l^{-1}) \tag{8}$$

representing a multiplicative application of a *base smoother*, M_l, in n_{st} repeated stages. For the $\mathrm{M_G}$ formulated in Section 3.3, the best performance is observed when M_l is chosen compatibly as an additive Schwarz preconditioner called the *contact-grain* smoother, or $\mathrm{M_{CG}}$ [26]. While other, black-box smoothers like Gauss-Seidel ($\mathrm{M_{GS}}$) and incomplete LU-factorization ($\mathrm{M_{ILU}}$) are possible, they either converge more slowly (often tenfold) or lead to the occasional stagnation of the Krylov solver. We therefore opt for M_l=$\mathrm{M_{CG}}$ in this work. We also note that a symmetric combination of $\mathrm{M_G}$ and $\mathrm{M_L}$ is possible [26], allowing M to be used in symmetric solvers like conjugate gradient.

3.2. Domain decomposition

To construct $\mathrm{M_G}$ and $\mathrm{M_L}$, Ω is first decomposed into N^g non-overlapping subdomains, Ω^{g_i}, referred to as *grain grids*. For this, a modified watershed segmentation algorithm proposed by [27] is applied to the image representing Ω. Fig.1b shows an example of such a decomposition for the pore-scale image in Fig.1a, where grain grids are depicted by the randomly colored regions. The interfaces shared between adjacent grain grids, Γ^{c_j}, are called *contact interfaces* and one is illustrated in Fig.1d. Watershed segmentation is a morphological operation in image analysis [6, 34] whose key feature is that Ω^{g_i} corresponds to a local enlargement of Ω, and Γ^{c_j} to a local constriction.

In addition to Ω^{g_i}, we construct a complementary set of N^{ζ} subdomains, Ω^{ζ_k}, called *contact grids*. Each contact grid covers an interface, Γ^{c_k}, plus a thin region around it. Fig.1c provides a visual schematic. To build Ω^{ζ_k}, successive morphological dilations, an operation in image analysis [48], of the pixels comprising Γ^{c_k} are performed. The thickness of Ω^{ζ_k} is proportional to the number of such dilations and can be adjusted by the user. Typically, a width of $\sim$12 pixels (6 per contact side) is sufficient. Notice that the union of contact grids does not cover Ω (i.e., $\Omega \neq \cup_k \Omega^{\zeta_k}$). Contact grids are only used by $\mathrm{M_L}$, and their function is to remove high-frequency errors that tend to concentrate near Γ^{c_j} after every application of $\mathrm{M_G}$. We note that contact grids in the PLMM preconditioner are allowed to overlap with each other, which is conceptually simpler and computationally more advantageous than the original geometric (non-algebraic) formulation of PLMM [27], wherein such grids had to be merged (see [26] for a discussion).

3.3. Global preconditioner

The global (or coarse) preconditioner $\mathrm{M_G}$ is defined as follows:

$$\mathrm{M_G^{-1}} = \hat{\mathrm{P}}\,(\hat{\mathrm{R}}\hat{\mathrm{A}}\hat{\mathrm{P}})^{-1}\hat{\mathrm{R}} \tag{9}$$

where $\hat{\mathrm{P}}$ and $\hat{\mathrm{R}}$ are the *effective prolongation* and *effective restriction* matrices, respectively.[1] Here, we set $\hat{\mathrm{R}} = \hat{\mathrm{P}}^{\top}$ and formulate $\hat{\mathrm{P}}$ as the multiplication of three matrices:

$$\hat{\mathrm{P}} = \mathrm{WQP} \tag{10}$$

We refer to W as the *permutation matrix*, to Q as the *reduction matrix*, and to P as the (reduced) *prolongation matrix*. Below, we describe the procedure for constructing each one, while referring the reader to [26] for further details.

Permutation (W). The permutation matrix, W, is square and consists of only 0 and 1 entries. Its function is to shuffle the columns of any matrix it right-multiplies. Hence, it is unitary, i.e., $\mathrm{WW}^{\top} = \mathrm{I}$. The shuffling is done in accordance with the domain decomposition in Section 3.2, such that the fine-grid entries associated with each grain grid Ω^{g_i}, and each contact interface Γ^{c_j}, are grouped together. Applying W to the linear system in Eq.6 yields:

$$\underbrace{\mathrm{W}^{\top}\hat{\mathrm{A}}\mathrm{W}}_{\mathrm{A}}\,\underbrace{\mathrm{W}^{\top}\hat{x}}_{x} = \underbrace{\mathrm{W}^{\top}\hat{b}}_{b} \quad \Rightarrow \quad \mathrm{A}x = b \tag{11a}$$

[1]Following the terminology introduced in [26], we use the term *effective* to distinguish $\hat{\mathrm{P}}$ from the prolongation matrix P introduced below.

where the permuted A, b, and x have the following block structures:

$$\mathrm{A} = \begin{bmatrix} \mathrm{A}^g_g & \mathrm{A}^g_c \\ \mathrm{A}^c_g & \mathrm{A}^c_c \end{bmatrix} \qquad b = \begin{bmatrix} b^g \\ b^c \end{bmatrix} \qquad x = \begin{bmatrix} x^g \\ x^c \end{bmatrix} \tag{11b}$$

$$\mathrm{A}^g_g = \begin{bmatrix} \mathrm{A}^{g_1}_{g_1} & \cdots & \mathrm{O} \\ \vdots & \ddots & \vdots \\ \mathrm{O} & \cdots & \mathrm{A}^{g_{N^g}}_{g_{N^g}} \end{bmatrix}_{N^f_g \times N^f_g} \qquad \begin{aligned} \mathrm{A}^c_g &= [\mathrm{A}^{c_i}_{g_j}]_{N^f_c \times N^f_g} \\ \mathrm{A}^g_c &= [\mathrm{A}^{g_i}_{c_j}]_{N^f_g \times N^f_c} \\ \mathrm{A}^c_c &= [\mathrm{A}^{c_i}_{c_j}]_{N^f_c \times N^f_c} \end{aligned} \qquad \begin{aligned} b^g &= [b^{g_i}]_{N^f_g \times 1} \\ b^c &= [b^{c_i}]_{N^f_c \times 1} \end{aligned} \tag{11c}$$

The super/subscripts g_i and c_j specify the entries/blocks that belong to either Ω^{g_i} or Γ^{c_j}, respectively. $N^f_{g_i}$ and $N^f_{c_j}$ are the number of fine-scale unknowns associated with Ω^{g_i} and Γ^{c_j}, respectively, and $N^f_g = \sum_i N^f_{g_i}$ and $N^f_c = \sum_j N^f_{c_j}$. Recall N^g is the total number of grain grids. The matrix A^g_g is square and block-diagonal, with square blocks $\mathrm{A}^{g_i}_{g_i}$, while A^g_c and A^c_g are thin and rectangular. Moreover, $\mathrm{A}^g_c = (\mathrm{A}^c_g)^\top$ and $\mathrm{A}^{g_i}_{c_j} = (\mathrm{A}^{c_j}_{g_i})^\top$ hold because of the self-adjoint nature of the PDEs in Eqs.1a and 2a and our choice to use Galerkin FEM to discretize them. Building W is trivial, thus cheap.

Reduction matrix (Q). The reduction matrix, Q, is square and consists of only 0 and 1 entries. Its function is to perform a column-sum, when right-multiplying a matrix, over all entries associated with each contact interface Γ^{c_j}. For the linear-elasticity PDE, this summation is done on a per coordinate-direction basis. Hence, Q is expressed as:

$$\mathrm{Q} = \begin{bmatrix} \mathrm{I}_{N^f_g \times N^f_g} & \mathrm{O} \\ \mathrm{O} & \mathrm{Q}^o \end{bmatrix} \qquad \mathrm{Q}^{\mathrm{o}} = \begin{bmatrix} \mathbf{1}^{c_1} & & \mathrm{O} \\ & \ddots & \\ \mathrm{O} & & \mathbf{1}^{c_N} \end{bmatrix}_{N^f_c \times N^o_c} \qquad \mathbf{1}^{c_i} = \begin{bmatrix} \mathrm{I}_{\omega \times \omega} \\ \vdots \\ \mathrm{I}_{\omega \times \omega} \end{bmatrix}_{N^f_{c_i} \times \omega} \tag{12}$$

where $N^o_c = N^c \omega$, with $\omega = 1$ for Eq.1 and $\omega = D$ for Eq.2. Recall N^c is the total number of contact interfaces. The parameter ω represents the number of degrees of freedom per fine grid (here, FEM node). Notice Q is block-diagonal, with its (1,1)-block an identity matrix and its (2,2)-block a block-diagonal matrix Q^{o} itself. Each block of Q^{o}, namely $\mathbf{1}^{c_i}$, consists of a series of vertically concatenated identity matrices of dimension $\omega \times \omega$. A symmetric application of Q to the permuted system in Eq.11a yields the reduced system below:

$$\mathrm{A}x = b\,, \quad x \simeq \mathrm{Q}x_M \quad \Rightarrow \quad \mathrm{Q}^\top \mathrm{A} \mathrm{Q} x_M = \mathrm{Q}^\top b \quad \Rightarrow \quad \mathrm{A}_{\mathrm{M}} x_M = b_M \tag{13a}$$

where A_{M} possesses the following block structure:

$$\mathrm{A}_{\mathrm{M}} = \begin{bmatrix} \mathrm{A}^g_g & \bar{\mathrm{A}}^g_c \\ \bar{\mathrm{A}}^c_g & \bar{\mathrm{A}}^c_c \end{bmatrix} \qquad \begin{aligned} \bar{\mathrm{A}}^c_g &= [\bar{\mathrm{A}}^{c_i}_{g_j}]_{N^o_c \times N^f_g} \\ \bar{\mathrm{A}}^g_c &= [\bar{\mathrm{A}}^{g_i}_{c_j}]_{N^f_g \times N^o_c} \\ \bar{\mathrm{A}}^c_c &= [\bar{\mathrm{A}}^{c_i}_{c_j}]_{N^o_c \times N^o_c} \end{aligned} \tag{13b}$$

The overbared blocks have smaller dimensions compared to those in Eq.11c (note N^f_c is replaced by N^o_c). The application of Q in Eq.13a simultaneously imposes an (integrated) flux balance across all Γ^{c_j} and a localization assumption that u or $\boldsymbol{u}$ are uniform along each Γ^{c_j}, both inherent to PLMM [26]. Building Q is trivial, thus cheap.

Prolongation matrix (P). The prolongation matrix, P, is tall and skinny with columns that define a coarse space wherein a close approximation to the solution, x_M, of the reduced system in Eq.13a exists. Each column of P is comprised of local solutions of the PDEs in Eqs.1 or 2 on one/two grain grids. Thus, P is sparse. It is built as follows:

$$\mathrm{P} = \begin{bmatrix} \mathrm{B} & \mathrm{C} \\ \mathrm{I} & \mathrm{O} \end{bmatrix}_{(N^f_g + N^o_c) \times (N^o_c + N^g)} \qquad \mathrm{B} = \begin{bmatrix} p^{g_1}_1 & p^{g_1}_2 & \cdots & p^{g_1}_n \\ p^{g_2}_1 & p^{g_2}_2 & \cdots & p^{g_2}_n \\ \vdots & \vdots & \ddots & \vdots \\ p^{g_m}_1 & p^{g_m}_2 & \cdots & p^{g_m}_n \end{bmatrix}_{N^f_g \times N^o_c} \qquad \mathrm{C} = \begin{bmatrix} c^{g_1} & & & \mathrm{O} \\ & c^{g_2} & & \\ & & \ddots & \\ \mathrm{O} & & & c^{g_m} \end{bmatrix}_{N^f_g \times N^g} \tag{14a}$$

The *basis matrix*, B, and *correction matrix*, C, are comprised of the following vectors:

$$c^{g_i} = (\mathrm{A}^{g_i}_{g_i})^{-1} b^{g_i} \tag{14b}$$

$$p^{g_i}_k = \begin{cases} -(\mathrm{A}^{g_i}_{g_i})^{-1} \bar{\mathrm{A}}^{g_i}_{c_j} \mathrm{R}^{c_j}_c e_k, & g_i \in G^{c_j} \quad c_j = \lceil k/\omega \rceil \\ O, & g_i \notin G^{c_j} \quad c_j = \lceil k/\omega \rceil \end{cases} \tag{14c}$$

$$e_k = [0, \cdots, \underbrace{0, 1, 0,}_{k-1,\, k,\, k+1} \cdots, 0]^{\top}_{N^o_c \times 1} \tag{14d}$$

where $p^{g_i}_k$ is referred to as a *basis vector*, and c^{g_i} as a *correction vector*, both defined on the grain grid Ω^{g_i}. In Eq.14a, we have used $n = N^o_c$ and $m = N^g$ for brevity. The unit vector e_k contains 1 in its k^{th} entry, with k corresponding to the contact interface with index $c_j = \lceil k/\omega \rceil$. The set G^{c_j} contains the indices of the only two grain grids that share Γ^{c_j}. According to Eq.14c, $p^{g_i}_k$ is non-zero only if Ω^{g_i} shares an interface Γ^{c_j} with another grain grid. Thus, B is sparse and only two of the basis vectors in each column of it are non-zero. Finally, the *contraction matrix*, $\mathrm{R}^{c_j}_c$, is defined as:

$$\mathrm{R}^{c_i}_c = \left[\Delta^{c_i}_{c_1}, \Delta^{c_i}_{c_2}, \cdots, \Delta^{c_i}_{c_{N^o_c}}\right]_{\omega \times N^o_c} \qquad \Delta^{c_i}_{c_j} = \begin{cases} \mathrm{I}_{\omega\times\omega} & \text{if } i = j \\ \mathrm{O}_{\omega\times\omega} & \text{if } i \neq j \end{cases} \tag{15}$$

Left-multiplying a $N^o_c \times 1$ vector defined on all contact interfaces (e.g., e_k) by $\mathrm{R}^{c_i}_c$ restricts it to a $\omega \times 1$ vector on Γ^{c_i}.

Building the prologation matrix P, outlined above is fully parallelizable, as it involves the calculation of $2N^o_c$ decoupled basis vectors and N^g decoupled correction vectors on non-overlapping subdomains (i.e., grain grids). Even so, computing P is the most computationally expensive step in constructing $\mathrm{M_G}$ via Eq.10 because it involves the repeated solution of Eq.1 or Eq.2 over each grain grid subject to different BCs. Since most c^{g_i} are zero herein, because the source terms f and $\boldsymbol{f}$ in Eqs.1-2 are set to zero ($c^{g_i} \neq 0$ only if Ω^{g_i} intersects the global boundary Γ^{ex}), the cost of building P is dominated by that of building B in Eq.14a. In Section 4, we propose a machine learning (ML) algorithm that significantly accelerates the construction of B, and thereby that of $\mathrm{M_G}$.

3.4. Local smoother

The high-frequency errors that remain after applying $\mathrm{M_G}$ tend to concentrate near contact interfaces. The compatible *contact-grain* smoother $\mathrm{M_{CG}}$, proposed by [26], specifically targets these errors by applying two additive schwarz preconditioners in immediate succession:

$$\mathrm{M}^{-1}_{\mathrm{CG}} = \mathrm{M}^{-1}_{\zeta} + \mathrm{M}^{-1}_{g}(\mathrm{I} - \hat{\mathrm{A}}\mathrm{M}^{-1}_{\zeta}) \tag{16}$$

The first, called the *contact-grid* smoother M_ζ, wipes out all errors within each contact grid (i.e., a small neighborhood around each contact interface), and the second, called the *grain-grid* smoother M_g, removes all errors inside each grain grid. Notice the multiplicative composition of M_ζ and M_g in Eq.16 resembles Eq.7. M_g and M_ζ take the standard algebraic forms common to all additive Schwarz preconditioners [9] below:

$$\mathrm{M}^{-1}_{g} = \sum_{i=1}^{N^g} \mathrm{E}^{g_i}_f (\mathrm{R}^{g_i}_f \hat{\mathrm{A}} \mathrm{E}^{g_i}_f)^{-1} \mathrm{R}^{g_i}_f\ , \quad \mathrm{M}^{-1}_{\zeta} = \sum_{i=1}^{N^\zeta} \mathrm{E}^{\zeta_i}_f (\mathrm{R}^{\zeta_i}_f \hat{\mathrm{A}} \mathrm{E}^{\zeta_i}_f)^{-1} \mathrm{R}^{\zeta_i}_f \tag{17}$$

The matrices $\mathrm{R}^{g_i}_f$ and $\mathrm{E}^{g_i}_f$ restrict and extend, respectively, any vector they left-multiply to/from Ω^{g_i} and Ω. Similarly, $\mathrm{R}^{\zeta_i}_f$ and $\mathrm{E}^{\zeta_i}_f$ restrict and extend to/from Ω^{ζ_i} and Ω. In an iterative solver, applying M_g entails solving N^g decoupled systems on grain grids, and applying M_ζ entails solving N^ζ decoupled systems on contact grids; all fully parallelizable.

4. Building the global preconditioner via machine learning

We propose a machine learning architecture, modified after ResUnet in [49], that yields the basis vectors, $p^{g_i}_k$, in Eq.14c at a much lower computational cost than solving them directly with a numerical solver. From Eq.14c and our discussions in Section 3.3, recall that for every contact interface Γ^{c_j}, two sets of basis vectors are computed: one on each of the two grain grids flanking Γ^{c_j}. Each set consists of ω basis vectors, where $\omega = 1$ for Poisson and $\omega = D$ for elasticity. This means that on every grain grid Ω^{g_i}, $\#C^{g_i} \times \omega$ basis vectors must be built, where $\#C^{g_i}$ is the number of contact interfaces intersecting $\partial\Omega^{g_i}$. Our ML algorithm aims to accelerate such repeated calculations on Ω^{g_i}.

4.1. Curating labeled training data

The inputs and outputs of the proposed ML algorithm consist of small *images* with dimensions 64×64 in 2D, and $48 \times 48 \times 48$ in 3D. These images contain the geometry of each grain grid and any fields defined on them, as exemplified by Fig.2 for the grain grid Ω^{g_1} with three contact interfaces Γ^{c_1}, Γ^{c_2}, and Γ^{c_3}. The outside of Ω^{g_1} is colored gray. The small image sizes of the subdomain, unlike whole-domain images required by other ML algorithms in the literature (e.g., [39]), ensure that training is rapid and the overhead in computer memory stays low. To capture a grain grid's geometry inside the above-prescribed dimensions, we first circumscribe the grain grid by its minimal bounding box, then crop it out of the pore-scale domain's original image. If this box is smaller/larger than the prescribed image dimensions, as it is almost always the case, we up/downsample it to match the required size. Upsampling is straightforward, as it involves cutting a pixel into sub-pixels then copying the pixel's value onto the sub-pixels. Downsampling requires mapping pixel values to those of a coarsened image, for which we use bi/trilinear interpolation.

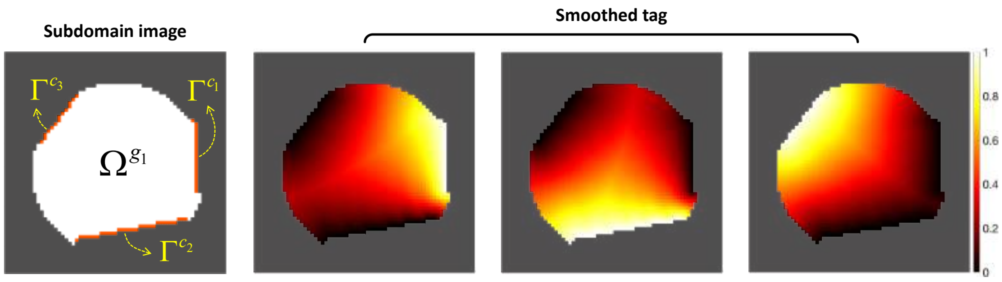

Figure 2: Example of a 2D subdomain (i.e., grain grid) image and its corresponding input features passed to the ML algorithm. The grain grid Ω^{g_1} consists of three contact interfaces Γ^{c_1}, Γ^{c_2}, and Γ^{c_3}. The smoothed tags are computed via Eq.18. From left to right, they correspond to basis vectors with BCs: (1) $u = 1$ on Γ^{c_1} and $u = 0$ on $\Gamma^{c_2} \cup \Gamma^{c_3}$; (2) $u = 1$ on Γ^{c_2} and $u = 0$ on $\Gamma^{c_1} \cup \Gamma^{c_3}$; and (3) $u = 1$ on Γ^{c_3} and $u = 0$ on $\Gamma^{c_1} \cup \Gamma^{c_2}$.

Before detailing the input feature required by the ML algorithm, let us first focus on what *output* we seek from it. Consider the Poisson problem defined on Ω^{g_1} in Fig.2. There are a total of three basis vectors we need to compute, corresponding to local BCs: (1) $u = 1$ on Γ^{c_1} and $u = 0$ on $\Gamma^{c_2} \cup \Gamma^{c_3}$; (2) $u = 1$ on Γ^{c_2} and $u = 0$ on $\Gamma^{c_1} \cup \Gamma^{c_3}$; and (3) $u = 1$ on Γ^{c_3} and $u = 0$ on $\Gamma^{c_1} \cup \Gamma^{c_2}$. Similarly for the 2D elasticity problem, there are six basis vectors in total, two per interface. For example, the two bases associated with Γ^{c_1} correspond to local BCs: (1) $\boldsymbol{u} = (1,0)$ on Γ^{c_1} and $\boldsymbol{u} = (0,0)$ on $\Gamma^{c_2} \cup \Gamma^{c_3}$; (2) $\boldsymbol{u} = (0,1)$ on Γ^{c_1} and $\boldsymbol{u} = (0,0)$ on $\Gamma^{c_2} \cup \Gamma^{c_3}$. The other four bases associated with Γ^{c_2} and Γ^{c_3} follow in a similar vein. We do not demand the ML algorithm to produce all basis vectors defined on Ω^{g_i} at once, but one at a time. This requires the input to not be a mere binary image of Ω^{g_i}, but *tagged* in some fashion to specify which basis vector of which interface we desire. This motivates us to now turn our attention to describing the input feature.

A naïve approach to crafting the input is to take each subdomain's image and assign integer labels to the various pixel types, e.g., interior of Ω^{g_i}, interface with non-zero BC, and interfaces with zero BCs. Unfortunately, this approach, as we have found, does not work because it leads to very slow (even non-convergent) training; also observed by [39]. Instead, we design a smooth input image with pixel values corresponding to a distance map defined by:

$$d(\boldsymbol{x}) = \frac{d(\boldsymbol{x}, \Gamma_0)}{d(\boldsymbol{x}, \Gamma_0) + d(\boldsymbol{x}, \Gamma_1)} \tag{18}$$

where Γ_1 is the interface at which a non-zero Dirichlet BC is imposed, and Γ_0 is the union of all other interfaces where the BCs are zero. For example, for the Poisson basis vector that corresponds to $u = 1$ on Γ^{c_1} and $u = 0$ on $\Gamma^{c_2} \cup \Gamma^{c_3}$ in Fig.2, we have $\Gamma_1 = \Gamma^{c_1}$ and $\Gamma_0 = \Gamma^{c_2} \cup \Gamma^{c_3}$. The $d(\boldsymbol{x}, \Gamma_0)$ and $d(\boldsymbol{x}, \Gamma_1)$ are Euclidean distances from the pixel position $\boldsymbol{x}$ to the closest pixel on Γ_0 and Γ_1, respectively. Fig.2 illustrates all three input features, hereafter referred to as *smoothed tags*, corresponding to the three basis vectors of the Poisson problem defined on Ω^{g_1}. Notice for the Poisson equation, both the ML input and output are either 64×64 images in 2D, or $48 \times 48 \times 48$ images in 3D.

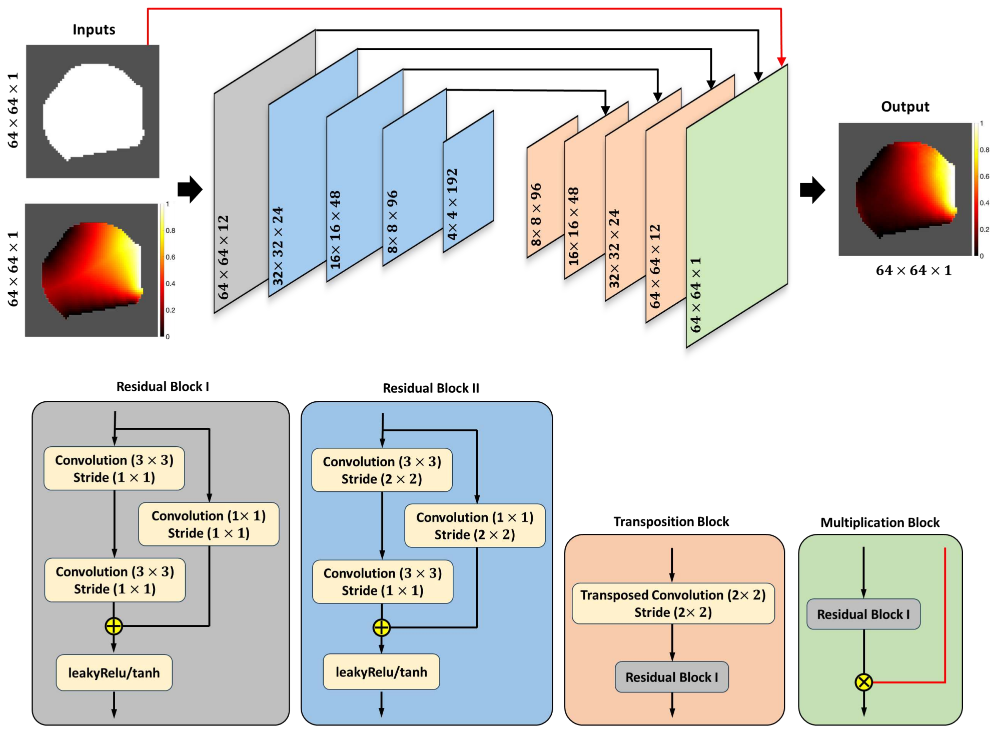

Figure 3: Proposed ML architecture, resembling a residual U-Net [49]. The top row shows the overall structure, which consists of encoder layers (gray and blue) and decoder layers (orange and green) connected by skip connections (black arrows). Each layer is a convolutional block with internal skip connections, as detailed by the sketches in the bottom row. The layers consist of Residual Blocks I and II, Transposition Blocks, and one Multiplication Block. The inputs to the ML algorithm are the image and smoothed tag of a subdomain (i.e., grain grid), and the output is the corresponding basis vector. The image is used (red arrow) by the Multiplication Block to filter out artifacts outside the subdomain's boundary.

For the elasticity problem, there are D bases associated with an interface, each a vector-valued function. Hence, the ML output is a $64 \times 64 \times D$ image in 2D, and a $48 \times 48 \times 48 \times D$ image in 3D, with the last dimension containing the D components of the basis vector's displacement field. To compute the D bases associated with each interface, we train D separate ML algorithms, one for each coordinate direction. For example, for the 2D subdomain Ω^{g_1} in Fig.2, two architectures are trained. When applied to Γ^{c_1}, both accept the associated smoothed tag (second image from the left in Fig.2) as input. The first architecture outputs the basis associated with the local BCs $\boldsymbol{u} = (1, 0)$ on Γ^{c_1} and $\boldsymbol{u} = (0, 0)$ on $\Gamma^{c_2} \cup \Gamma^{c_3}$, while the second outputs the basis associated with $\boldsymbol{u} = (0, 1)$ on Γ^{c_1} and $\boldsymbol{u} = (0, 0)$ on $\Gamma^{c_2} \cup \Gamma^{c_3}$.

To curate training data, we generate random disk packs in 2D and random sphere packs in 3D; like the one shown in Fig.1a. Each domain is decomposed via the watershed-based algorithm described in Section 3.2 into subdomains. After cropping and resizing the subdomain images, per the up/downsampling procedure already discussed, we compute smoothed tags via Eq.18 for all bases of each subdomain. We next compute the basis vectors themselves using a FEM solver on the cropped (but unresized) subdomain images, then resize them to match the dimensions of the smoothed tags. The smoothed tags and basis vectors are paired to form a labeled dataset. In computing the bases for the elasticity problem, we set the Lamé parameters to $\lambda = 8.3$ GPa and $\mu = 44.3$ GPa, corresponding to α-quartz [50]. Later we demonstrate the generalizability of the trained ML algorithms, not only to subdomain geometries other than those of disk/sphere packs, but also other stiffness tensors. The datasets for Poisson and elasticity consist, separately, of 8,000 data points, each a triplet of a binary image, smoothed tag, and a basis vector. The binary image assigns 1 to pixels belonging to the subdomain, and 0 to all other pixels. But if downscaled, this image becomes grayscale with pixels along the subdomain's boundary assuming values between 0 and 1. Finally, we randomly split the dataset into 5,600 for training, 1,600 for validation (to tune hyperparameters and prevent overfitting), and 800 for testing.

4.2. *Machine learning architecture*

Fig.3 illustrates the ML architecture we train to predict the basis vector (output) corresponding to a given smoothed tag and subdomain image (inputs). For the elasticity problem, D such algorithms are trained separately to predict the D basis vectors associated with each smoothed tag (or contact interface). The architecture in Fig.3 resembles a residual U-Net [49], consisting of several encoder layers (gray and blue) and decoder layers (orange and green) linked by skip connections (black arrows). Each layer is a convolutional block with internal skip connections, as sketched in Fig.3. The layers consist of Residual Blocks I and II, Transposition Blocks, and a Multiplication Block. The input subdomain image is used (red arrow) by the Multiplication Block to filter out artifacts outside the subdomain's boundary. The activation function used in Residual Blocks I and II is `leakyRelu` for the Poisson problem, but `tanh` for the elasticity problem. The latter is selected because the displacement variable $\boldsymbol{u}$ in elasticity can assume both negative and positive values in a basis vector, whereas the Poisson variable u is guaranteed to be always positive. Fig.3 also annotates the size of the convolutional stencils, their stride lengths, and each layer's input/output dimensions.

We define the loss functions of the Poisson, L_p, and elasticity, L_e, problems as follows:

$$L_p = \frac{1}{N}\sum_{i=1}^{N}(y_{true} - y_{pred})^2 + \alpha_p \frac{1}{N}\sum_{i=1}^{N}(\Delta y_{pred} + f)^2 \tag{19a}$$

$$L_e = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_{true} - \boldsymbol{y}_{pred})^2 + \alpha_e \frac{1}{N}\sum_{i=1}^{N}(\nabla\cdot\boldsymbol{\sigma}(\boldsymbol{y})_{pred} + \boldsymbol{f})^2 \tag{19b}$$

where the first term measures the *data mismatch* between the true (i.e., solver-computed) and predicted (i.e., ML-computed) basis vectors. The second term is the *physics mismatch*, or norm of the PDE's residual, in the predicted basis vectors. The weights α_p and α_e control the relative importance of the data- versus physics-based losses. We implemented the ML architecture in Fig.3 using MATLAB's Deep Learning Toolbox and trained it on a machine with an NVIDIA GeForce GTX 1660 Super graphics card. The batch size was set to 40, learning rate to 10^{-5}, and training was allowed to progress for 200 epochs. Early stoppage was used as the mechanism to prevent overfitting.

4.3. *Enforcing partition of unity on bases*

The basis vectors defined on a subdomain, Ω^{g_i}, must satisfy an important constraint: *partition of unity*. For the Poisson equation, this means that if we sum all the bases on Ω^{g_i}, the result must be an all-ones function:

$$\sum_{\forall c_j \in C^{g_i}} \varphi_{c_j}^{g_i} = 1 \tag{20}$$

For emphasis and notational simplicity, we have used $\varphi_{c_j}^{g_i}$, instead of $p_k^{g_i}$, in Eq.20 to denote the basis vector associated with grain grid Ω^{g_i} and contact interface Γ^{c_j}. The set C^{g_i} contains the indices of all interfaces intersecting $\partial\Omega^{g_i}$.

For elasticity, this equation takes the form:

$$\sum_{\forall c_j \in C^{g_i}} \boldsymbol{\varphi}_{c_j d}^{g_i} = \mathbf{1}_d \tag{21}$$

where we have again used the simpler notations $\varphi_{c_j x}^{g_i}$ and $\varphi_{c_j y}^{g_i}$ (and $\varphi_{c_j z}^{g_i}$ in 3D) to denote the D basis vectors associated with Ω^{g_i} and Γ^{c_j}. Here, $\mathbf{1}_d$ is a $D \times 1$ constant vector field with 1 for its d^{th} component and 0 for its other components. We have found that if Eqs.20 and 21 are not enforced explicitly on the ML-predicted basis vectors, during a postprocessing step, the performance of the ML-preconditioned Krylov solver in Section 6 deteriorates greatly.

To impose Eq.20, we simply divide (in pointwise fashion) each ML-predicted basis by the sum of all bases on Ω^{g_i}. To impose Eq.21, we first divide the d^{th} component of each ML-predicted basis by the sum all the d-components of all the bases on Ω^{g_i}. This ensures the normalized d-components of all bases sum to one. Next, we normalize the other ($\neq d$) components by subtracting their arithmetic mean over all the bases from that of each individual basis (also in pointwise fashion). This ensures that the non-d-components of the normalized bases sum to zero.

4.4. Smoothing machine learned bases

The ML-predicted bases tend to be dominated by high-frequency errors that exhibit a checkerboard-like pattern (shown later in Figs.5-6). While in Section 6 we report that such errors have negligible impact on the overall cost of the Krylov solver, we propose an iterative strategy to arbitrarily improve the accuracy of ML-predicted bases, and thereby the ML-built $\mathrm{M_G}$, if so desired. The approach requires performing a small number of iterations with a Gauss-Seidel (or any other) smoother that rapidly attenuates high-frequency errors. The lower-triangular matrix used to perform the local Gauss-Seidel iterations on Ω^{g_i} is derived directly by restricting the global matrix $\hat{\mathrm{A}}$ in the linear system Eq.6 onto Ω^{g_i}. Each smoothing iteration is cheap and equivalent to a non-trainable convolution layer appended to the end of the ML architecture in Fig.3 (see [51]). Since the accuracy of the ML-predicted basis vectors varies across subdomains, one can ensure quality control by adapting the number of smoothing iterations such that the relative error:

$$E_{\mathrm{ML}} = \frac{|\, ||R_i|| - ||R_{i-1}||\, |}{||R_0||} \tag{22}$$

satisfies a desired tolerance, T_{ML}. R_0 and R_i are local residuals on Ω^{g_i} at the 0th and ith smoothing steps. The smaller a tolerance we impose on E_{ML}, the more iterations are required, and the more costly the basis calculations become.

5. Problem set

To test the performance of the ML-built two-level preconditioner, M, in solving the Poisson and elasticity problems, we consider the 2D and 3D porous microstructures shown in Fig.4. They consist of a 2D disk pack (P2D), a 2D sandstone (S2D) [52], a 3D sphere pack (P3D), and a 3D bone specimen (BONE) [53]. Each domain is decomposed into grain grids and contact grids via the watershed segmentation algorithm described in Section 3.2, and they are illustrated by the randomly colored regions in Fig.4. Table 1 further summarizes each domain's image size, physical dimensions, number of FEM elements and nodes, number of grain grids N^g, and number of contact grids N^ζ.

With reference to Fig.4, we impose the following BCs on the domains. For the Poisson equation in 2D, we set $u=2$ on the left ($x=0$) and $u=0$ on the right ($x=L_x$) side of each domain. In 3D, we set $u=2$ and $u=0$ on the top ($z=L_z$) and bottom ($z=0$) sides, respectively. All lateral boundaries in 2D/3D are flux-free (i.e., homogeneous Neumann). For the elasticity equation in 2D, $\boldsymbol{u} = (-1,0)$ and $\boldsymbol{u} = (0,0)$ are set on the left and right boundaries, respectively. In 3D, $\boldsymbol{u} = (0,0,0)$ is imposed on the top side and $\boldsymbol{u} = (0,0,-1)$ on the bottom side. All lateral boundaries in 2D/3D are stress-free. The Lamé parameters of all domains are $\lambda=8.3$ GPa and $\mu=44.3$ GPa, same as those used to train the ML algorithms in Section 4. In Appendix C, we demonstrate the algorithms' transferability to other stiffnesses.

We solve the linear system, Eq.6, associated with the Poisson and elasticity problems for each domain with a right-preconditioned GMRES solver. The preconditioners probed herein are: (1) the two-level PLMM preconditioner M, whose coarse preconditioner $\mathrm{M_G}$ is built by a numerical solver; (2) the same two-level preconditioner M, except whose coarse preconditioner $\mathrm{M_G}$ is built by the trained ML algorithms of Section 4; and as benchmark, (3) an AMG preconditioner [54] constructed entirely by a numerical solver. To distinguish between preconditioners (1) and (2), we refer to them hereafter as $\mathrm{M_{SOL}}$ and $\mathrm{M_{ML}}$, respectively. As discussed in Section 3.1, $\mathrm{M_{SOL}}$ and $\mathrm{M_{ML}}$ require a local smoother, for which we use Eq.8 with the base smoother M_l=$\mathrm{M_{CG}}$ and number of smoothing stages $n_{st} = 1$; following recommendations in [26]. Unlike $\mathrm{M_{SOL}}$ and $\mathrm{M_{ML}}$, the AMG preconditioner is multilevel, with the number of levels (>2) determined automatically and summarized in Appendix A. Per custom, one pre- and one post-smoothing operation is performed in AMG via Gauss-Seidel per level. We declare GMRES to have "converged" if the normalized residual satisfies $||\hat{A}\hat{x} - \hat{b}||/||\hat{b}|| < 10^{-9}$ or the number of iterations reaches 500. All simulations are run in *series*.

Table 1: Geometric and fine/coarse-grid properties of the domains in Fig.4, used to test the ML-built preconditioner proposed.

	Image pixels	Domain size (mm)	FEM elements	FEM nodes	Grain grids (N^g)	Contact grids (N^ζ)
P2D	4,000× 4,000	40×40	12,750,317	12,866,447	480	688
S2D	4,000×4,000	40×40	12,704,770	12,775,648	227	383
P3D	150×150×300	1.5×1.5×3	3,971,225	4,506,890	581	1,040
BONE	150×150×300	1.5×1.5×3	4,677,385	5,102,232	319	1,255

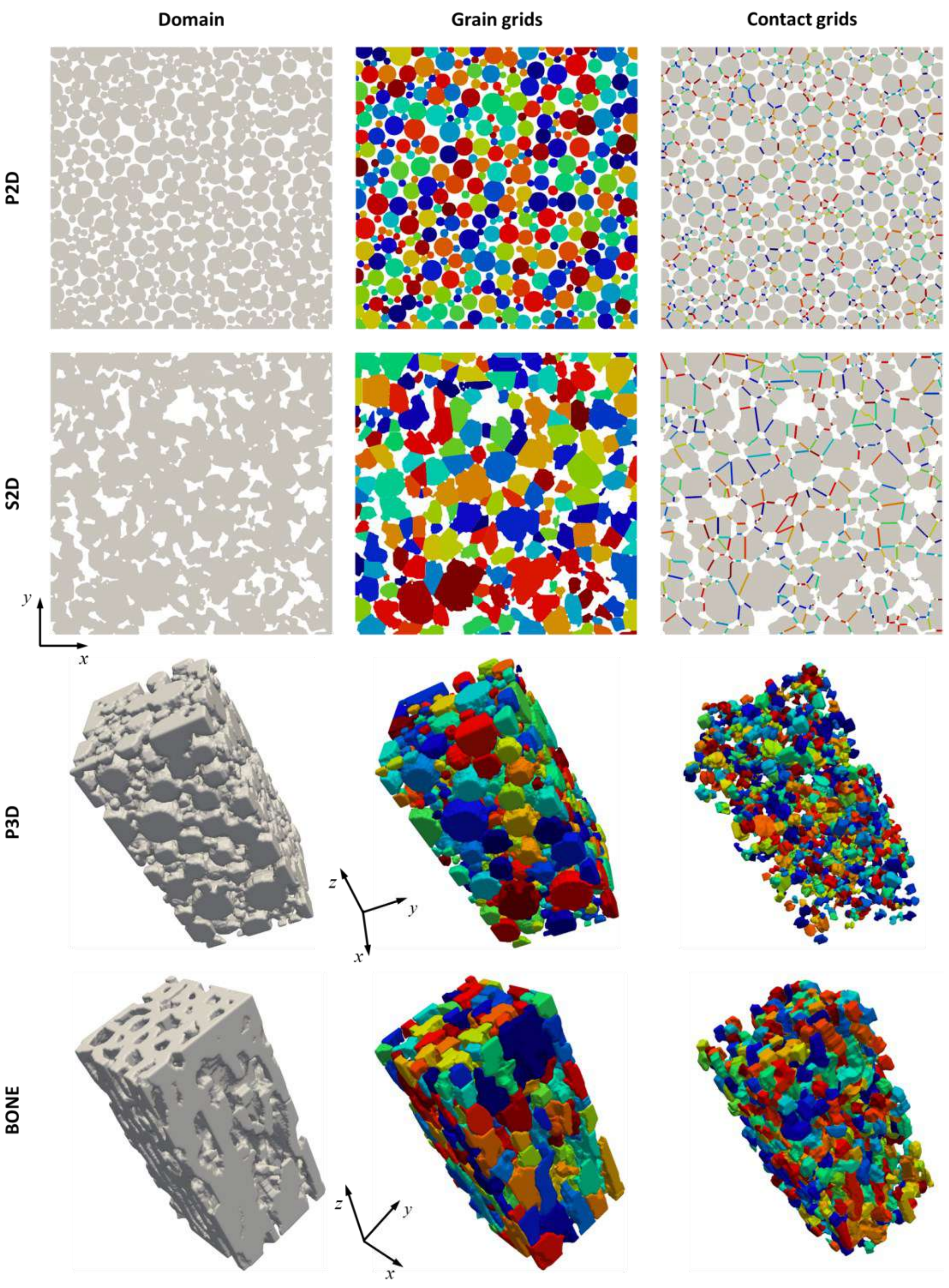

Figure 4: Porous geometries used to test the ML-accelerated multiscale preconditioner for the Poisson and elasticity equations. From top to bottom, they include a 2D disk pack (P2D), a 2D sandstone (S2D), a 3D sphere pack (P3D), and a 3D bone specimen (BONE). From left to right, each domain's geometry and corresponding grain grids and contact grids are illustrated. The last two are obtained from the decomposition algorithm described in Section 3.2 and are depicted as randomly colored regions.

6. Results

We present results in two parts. In Section 6.1, we probe the accuracy of the trained ML algorithms in constructing basis vectors on subdomains. Since the algorithms in Section 4 are trained on disk/sphere packs, we consider both in-distribution (unseen disk/sphere packs) and out-of-distribution (S2D and BONE) subdomains. For the elasticity problem only, we also probe the transferrability of the ML algorithm to Poisson's ratios different from that used during training (i.e., $\nu \neq 0.08$). Next in Section 6.2, we compare the convergence rates and wall-clock times (WCT) of the $\mathrm{M_{SOL}}$, $\mathrm{M_{ML}}$, and AMG preconditioners applied within GMRES in solving the Poisson and elasticity equations.

6.1. Basis vectors built by machine learning

6.1.1. In-distribution subdomains

Using the notation introduced in Section 4.3 for the basis vectors (i.e., $\varphi^{g_i}_{c_j}$ and $\varphi^{g_i}_{c_j d}$ with $d \in \{x, y, z\}$), Figs.5 and 6 compare basis vectors predicted by the trained ML algorithms against those obtained from a numerical solver for the Poisson and elasticity problems, respectively. The associated subdomain images and smoothed tags are also shown, which are chosen from the testing dataset (i.e., unseen samples) defined in Section 4.1. The ML bases are in good agreement with those from the solver, save for high-frequency, checkerboard error patterns that appear to be typical of convolutional neural networks (see [45]). The ML algorithms were trained while setting $\alpha_p = \alpha_e = 0$ in Eq.19.

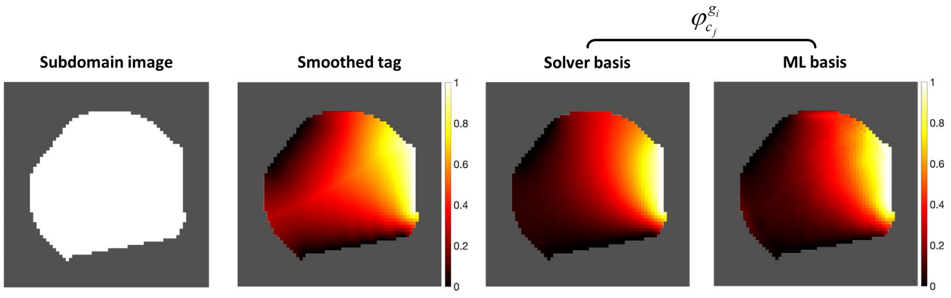

Figure 5: Schematic of a subdomain image, one of its smoothed tags, and the corresponding in-distribution basis vector for the Poisson problem. Two basis vectors are shown, one computed using a numerical solver and another predicted by the trained ML algorithm. The basis is denoted by $\varphi^{g_i}_{c_j}$, instead of $p^{g_i}_k$, to emphasize that it corresponds to the grain grid Ω^{g_i} and interface Γ^{c_j}. Given the annotations in Fig.2, $i=1$ and $j=$‘ here.

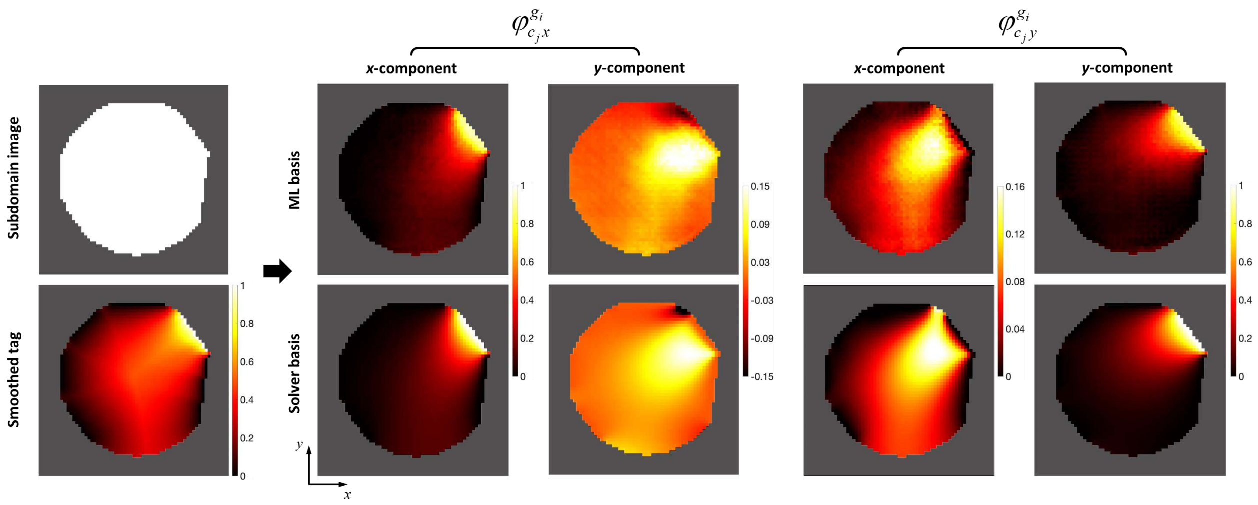

Figure 6: Schematic of a subdomain image, one of its smoothed tags, and the two corresponding in-distribution basis vectors for the Elasticity problem. The bottom row shows basis vectors computed using a numerical solver, and the top row bases predicted by the trained ML algorithms. Notice two basis vectors are associated with each interface, denoted by $\varphi^{g_i}_{c_j x}$ and $\varphi^{g_i}_{c_j y}$, instead of $p^{g_i}_k$, to emphasize they correspond to the grain grid Ω^{g_i}, interface Γ^{c_j}, and a non-zero Dirichlet BC imposed along the x or y coordinate direction. $\varphi^{g_i}_{c_j x}$ and $\varphi^{g_i}_{c_j y}$ are output by separate ML algorithms.

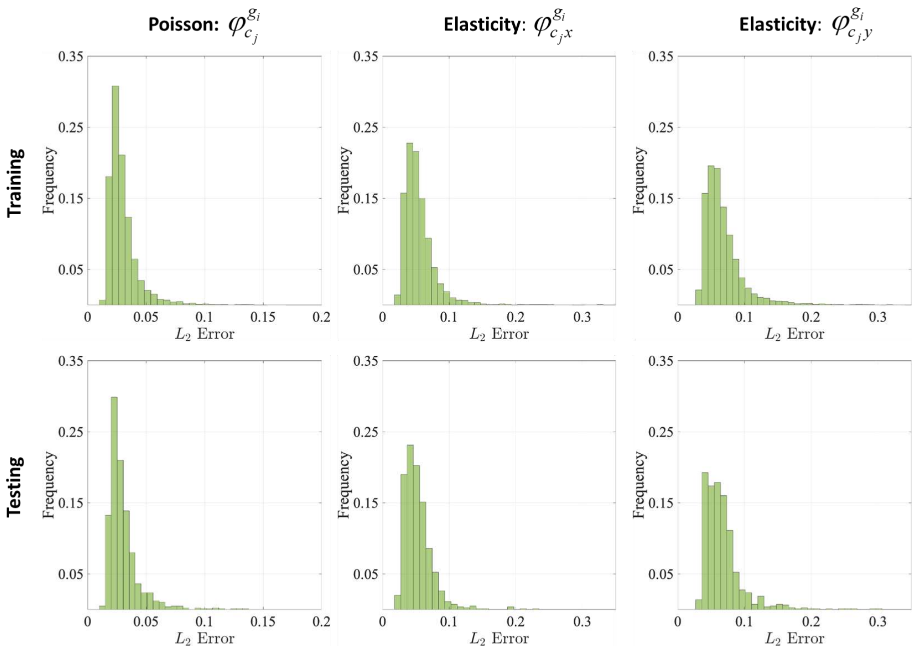

Figure 7: PDFs of L_2-errors associated with the ML-predicted basis vectors for the Poisson and elasticity problems defined on subdomains of 2D disk packs. The top row shows training errors, and the bottom row testing errors. Most errors are <5% for Poisson and <10% for elasticity.

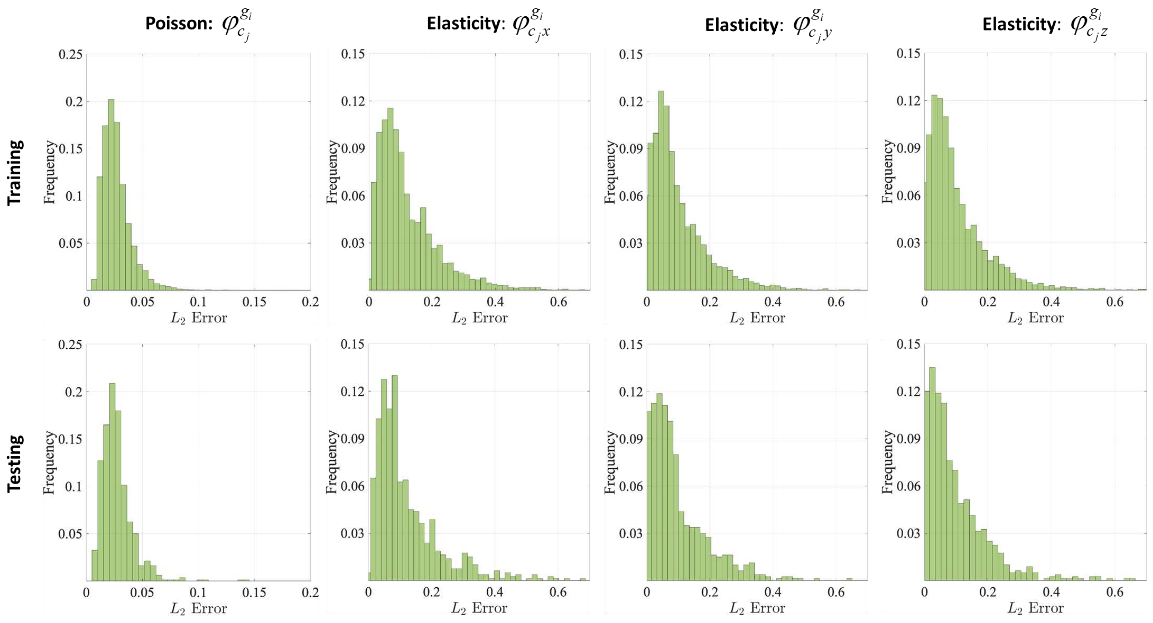

Figure 8: PDFs of L_2-errors associated with the ML-predicted basis vectors for the Poisson and elasticity problems defined on subdomains of 3D sphere packs. The top row shows training errors, and the bottom row testing errors. Most errors are <5% for Poisson and <20% for elasticity.

344 To quantify the L_2-errors between the ML- and solver-computed basis vectors, we define:

$$E_2^{\chi} = \left(\frac{1}{|\Omega^{g_i}|} \int_{\Omega^{g_i}} \|\chi_{ml} - \chi_{solver}\|^2 \mathrm{d}\Omega \right)^{1/2} \Big/ \ \sup_{\Omega^{g_i}} \|\chi_{solver}\| \qquad (23)$$

where χ is a placeholder for either u in Poisson or $\boldsymbol{u}$ in elasticity. χ_{ml} and χ_{solver} denote the solver- and ML-predicted basis vectors, respectively. The PDFs of L_2-errors so obtained for all subdomains in the training and testing datasets are shown in Figs.7-8 for the Poisson and elasticity problems. Fig.7 corresponds to basis vectors defined on subdomains of 2D disk packs, whereas Fig.8 corresponds to basis vectors on subdomains of 3D sphere packs. Given that training and testing datasets both consist of disk/sphere-pack subdomains, Figs.7-8 constitute *in-distribution* errors.

We see that testing errors for unseen samples are comparable to the training errors, largely <5% for the 2D Poisson and <10% for the 2D elasticity problem. In 3D, testing errors are <5% for Poisson and <20% for elasticity. The higher 3D errors are likely due to the fact that even though an equal number of data points (i.e., 5,600) were used to train both ML algorithms in 2D and 3D, each 3D subdomain has on average a larger number of contact interfaces, hence basis vectors. Therefore, the 3D training set covers a smaller range of variability in subdomain geometries than the 2D set. Augmenting the training set in the future could reduce the 2D and 3D testing errors further. Finally, recall the ML algorithms in Figs.5-8 were trained in a purely data-driven fashion, with $\alpha_p = \alpha_e = 0$ in the loss functions given by Eq.19. In Appendix B, we set $\alpha_p = 1$ and 10 for the Poisson equation and show that including the PDE's residual in the loss function neither improves training speed nor the accuracy of the predicted bases in any significant way.

6.1.2. Out-of-distribution subdomains

A key question we wish to answer here is whether the trained ML algorithms in Section 4 on disk/sphere packs also apply to subdomains of other, *out-of-distribution*, geometries without having to be retrained? Figs.9 and 10 suggest the answer is *yes*. Comparing the ML- and solver-built basis vectors for the Poisson and elasticity problems on two subdomains of the S2D domain shows very good agreement. Figs.11-12 also depict the PDFs of L_2-errors, obtained

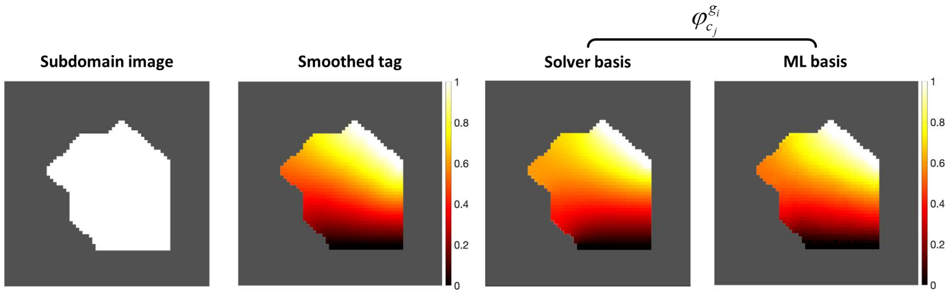

Figure 9: Schematic of a subdomain image, one of its smoothed tags, and the corresponding out-of-distribution basis vector for the Poisson problem. Two basis vectors are shown, one computed using a numerical solver and another predicted by the trained ML algorithm.

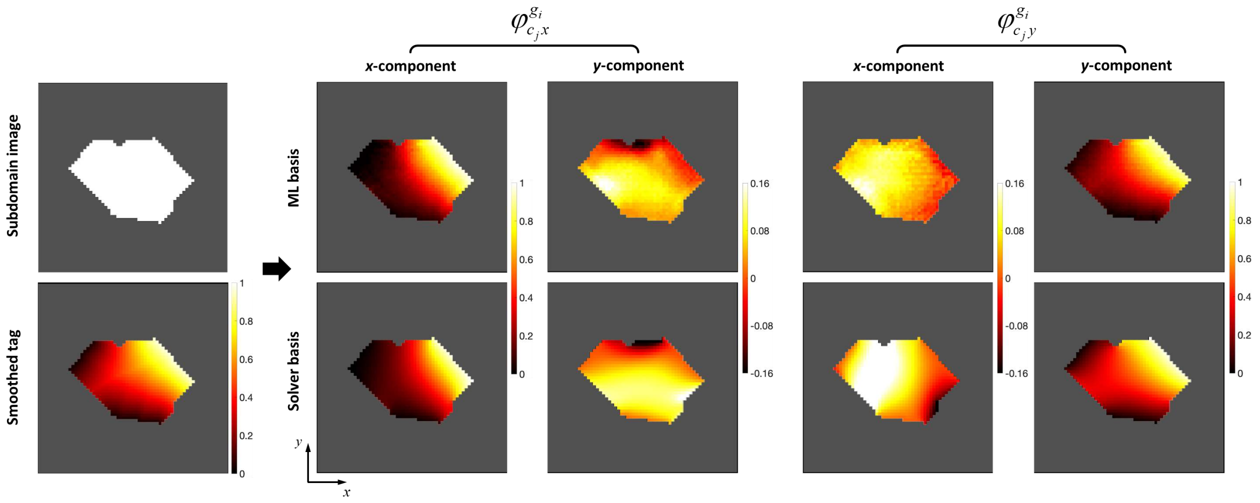

Figure 10: Schematic of a subdomain image, one of its smoothed tags, and the two corresponding out-of-distribution basis vectors for the Elasticity problem. The bottom row shows basis vectors computed using a numerical solver, and the top row bases predicted by the trained ML algorithms.

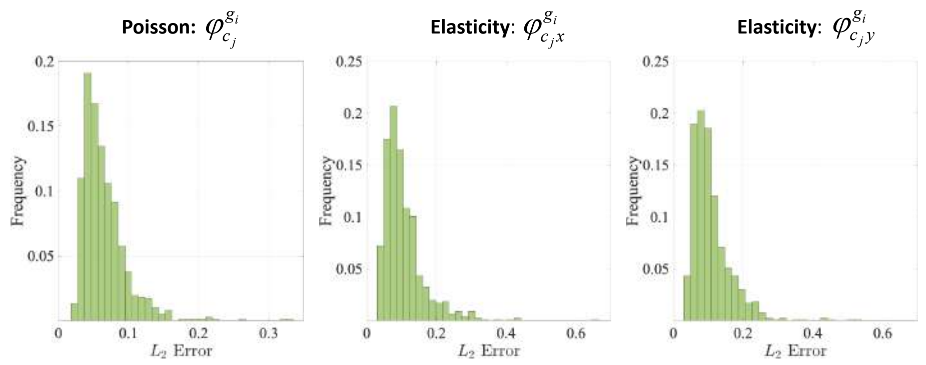

Figure 11: PDFs of L_2-errors associated with the ML-predicted basis vectors for the Poisson and elasticity problems defined on subdomains of the S2D domain. Most of these out-of-distribution errors are <10% for Poisson and <20% for elasticity.

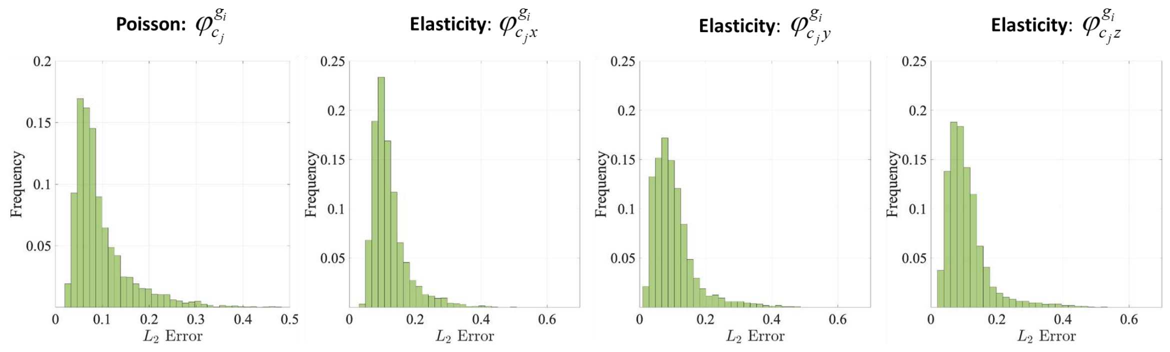

Figure 12: PDFs of L_2-errors associated with the ML-predicted basis vectors for the Poisson and elasticity problems defined on subdomains of the BONE domain. Most of these out-of-distribution errors are <20% for Poisson and <20% for elasticity.

via Eq.23, for all the ML-built basis vectors of the Poisson and elasticity equations on S2D and BONE subdomains.

These out-of-distribution errors are slightly larger than the in-distribution errors (computed on the testing dataset) in Figs.7-8, often by a factor of ∼2. Specifically, errors are <10% for Poisson and <20% for elasticity in S2D, and <20% for Poisson and <20% for elasticity in BONE. Considering no representative subdomains of either geometry were included in the training dataset, the accuracy of the ML algorithms is encouraging. Finally, recall that the ML-algorithms for the elasticity problem used in Figs.6 and 7-8 were trained in Section 4 on data that assumed a single Poisson's ratio of $\nu = 0.08$; corresponding to α-quartz. In Appendix C, we demonstrate these trained algorithms also apply to other values of ν without having to retrain them, with most L_2-errors <20%. In the next section, we assess the performance of the coarse preconditioner, $\mathrm{M_G}$, built from the above ML-predicted basis vectors.

6.2. Two-level preconditioners built by machine learning

6.2.1. Accuracy of the first-pass solution

The coarse preconditioner, $\mathrm{M_G}$, outlined in Section 3.3 can be used to obtain an approximate solution via $\hat{x}_{aprx} = \mathrm{M_G}^{-1}\hat{b}$. We call $\hat{x}_{aprx}$ the *first-pass* solution and compute it with $\mathrm{M_G}$'s constructed by: (1) a numerical solver ($\mathrm{M_{G,SOL}}$); (2) the trained ML algorithms of Section 4 with *no* smoothing iterations performed on the basis vectors ($\mathrm{M_{G,ML}}$, $T_{\mathrm{ML}} = \infty$); and (3) the trained ML algorithms of Section 4 *with* smoothing iterations performed on the basis vectors ($\mathrm{M_{G,ML}}$, $T_{\mathrm{ML}} < \infty$). In (3), iterations are performed until $E_{\mathrm{ML}} < T_{\mathrm{ML}}$ in Eq.22, where T_{ML} is a user-defined tolerance. The $\mathrm{M_G}$ in (2) can thus be viewed as corresponding to $T_{\mathrm{ML}} = \infty$, which we adopt hereafter to mean "no basis smoothing."

Figs.13-15 compare the first-pass solutions obtained from the above $\mathrm{M_G}$'s against the exact solution. Fig.13 shows the spatial distributions of u in the Poisson equation for all domains, where a very good agreement between all first-pass solutions and the exact solution is seen. Specifically, the u from ML with $T_{ML} = \infty$ is impressively accurate. Fig.14 shows the spatial distributions of $\boldsymbol{u} = (u_x, u_y)$ in the elasticity equation for the 2D domains, P2D and S2D.

The 3D domains, P3D and BONE, are shown in Fig.15, where only the axial u_z and radial $(u_x^2 + u_y^2)^{1/2}$ displacements of $\boldsymbol{u} = (u_x, u_y, u_z)$ are illustrated for brevity. Once again, very good agreement between the first-pass and exact solutions are observed, however this time *only in the axial direction* (i.e., u_x in 2D, and u_z in 3D). In the lateral/radial directions, the displacement component from $M_{G,ML}$ with $T_{ML} = \infty$ agrees rather poorly with that of the exact solution. Performing basis smoothing iterations in $M_{G,ML}$ with $T_{ML} = 10^{-5}$ improves the accuracy of lateral/radial component and brings it close to that of $M_{G,SOL}$. However, notice the first-pass solution from $M_{G,SOL}$ incurs some errors itself, to which the first-pass solution from $M_{G,ML}$ can only asymptote to in the limit $T_{ML} \to 0$, but never surpass.

Table 2 lists the relative L_2-errors of the first-pass solutions of the Poisson and elasticity equations computed via $M_{G,SOL}$ and $M_{G,ML}$ with $T_{ML} = \infty$, 1, 10^{-2}, 10^{-4}, and 10^{-9} on all the domains. The errors are expressed as percentages and computed via Eq.23, except with Ω^{g_i} replaced by Ω. First, notice that even without basis smoothing ($T_{ML} = \infty$), the $M_{G,ML}$ approximation has a very low error (<1% in 2D and <5% in 3D) and comparable to that of $M_{G,SOL}$. This renders the first-pass solution of $M_{G,ML}$ with $T_{ML} = \infty$ useful in a wide range applications where tolerance for error is moderate-to-high (e.g., subsurface engineering). Second, Table 2 suggests that basis smoothing iterations do not necessarily improve the accuracy of ML-predicted first-pass solutions unless a very large number is performed. Specifically, we see improvement for $T_{ML} = 10^{-9}$ where $M_{G,ML}$ and M_{SOL} have identical accuracy. But as mentioned earlier, such a small T_{ML} comes at a very high cost and is not recommended. In the next section, we show that despite the negligible impact on first-pass solutions, basis smoothing noticeably improves convergence in Krylov solvers.

Table 2: L_2-errors (%) for the first-pass solutions of the Poisson and elasticity problems obtained from a single application of the coarse preconditioner M_G built via a numerical solver ($M_{G,SOL}$) and ML algorithm ($M_{G,ML}$) with different basis-smoothing tolerances ($E_{ML} < T_{ML}$ in Eq.22).

		$M_{G,SOL}$	$M_{G,ML}$, $T_{ML} = \infty$	$M_{G,ML}$, $T_{ML} = 1$	$M_{G,ML}$, $T_{ML} = 10^{-2}$	$M_{G,ML}$, $T_{ML} = 10^{-4}$	$M_{G,ML}$, $T_{ML} = 10^{-9}$
Poisson	P2D	0.01	0.05	0.04	0.07	0.12	0.01
	S2D	0.03	0.09	0.07	0.10	0.20	0.03
	P3D	1.08	1.69	1.41	2.34	2.31	1.08
	BONE	2.74	3.09	2.84	2.84	2.80	2.74
Elasticity	P2D	0.27	0.45	0.44	0.41	0.37	0.31
	S2D	0.09	0.31	0.27	0.42	0.49	0.09
	P3D	3.34	5.15	4.67	4.37	3.77	3.35
	BONE	1.95	6.28	4.42	2.90	2.30	1.95

6.2.2. Convergence rate of the Krylov solver

Fig.16 plots the normalized residual ($||\hat{A}\hat{x} - \hat{b}||/||\hat{b}||$) versus the number of GMRES iterations preconditioned by AMG, M_{SOL}, and M_{ML} with $T_{ML} = \infty$, 10^{-2}, and 10^{-4} for the Poisson and elasticity equations defined on the P2D, S2D, P3D and BONE domains. Recall M_{SOL} and M_{ML} are obtained by combining the coarse preconditioners $M_{G,SOL}$ and $M_{G,ML}$ from the previous section with the contact-grain smoother defined in Section 3.4 using Eq.7 (i.e., M_L=M_{CG}). Three key observations stand out: (1) The convergence rate of M_{ML} with $T_{ML} = \infty$ is almost indistinguishable from M_{SOL} in the Poisson problem, and only slightly slower than M_{SOL} in the elasticity problem. This is good news, as it indicates the ML-built M_G is as good as the solver-built one. Therefore, we recommend $T_{ML} = \infty$ due to its lower cost; (2) As $T_{ML} \to 0$, the convergence rate of M_{ML} improves noticeably and eventually asymptotes to that of M_{SOL}. Recall this is contrary to Table 2, where reducing T_{ML} had a negligible impact on the first-pass solutions, except at very small T_{ML}; (3) In all cases, GMRES preconditioned by M_{SOL} or M_{ML} converges much faster than AMG, especially in the elasticity problem. We remark that the checkerboard errors in the ML-predicted bases (Fig.10) minimally affect the convergence rate of GMRES preconditioned by M_{ML} with $T_{ML} = \infty$, because such high-frequency errors are wiped out by the smoother M_{CG} integrated into M_{ML}. We next discuss the wall-clock times (WCTs) of the above simulations.

6.2.3. Computational cost

Figs.17 and 18 depict the wall-clock times (WCTs) in seconds associated with building the coarse preconditioner M_G and solving the linear system via GMRES to satisfy $||\hat{A}\hat{x} - \hat{b}|| \,/\, ||\hat{b}|| < 10^{-9}$ for the Poisson and elasticity equations, respectively, defined on all the domains. The total cost, including that of setting up the smoother in M_{SOL} and M_{ML}, is also shown. The smoother setup involves performing an LU decomposition of all local systems in Eq.17 defined on Ω^{g_i} and Ω^{ξ_k}. All WCTs are plotted versus the tolerance (T_{ML}) used for basis smoothing iterations in the building of

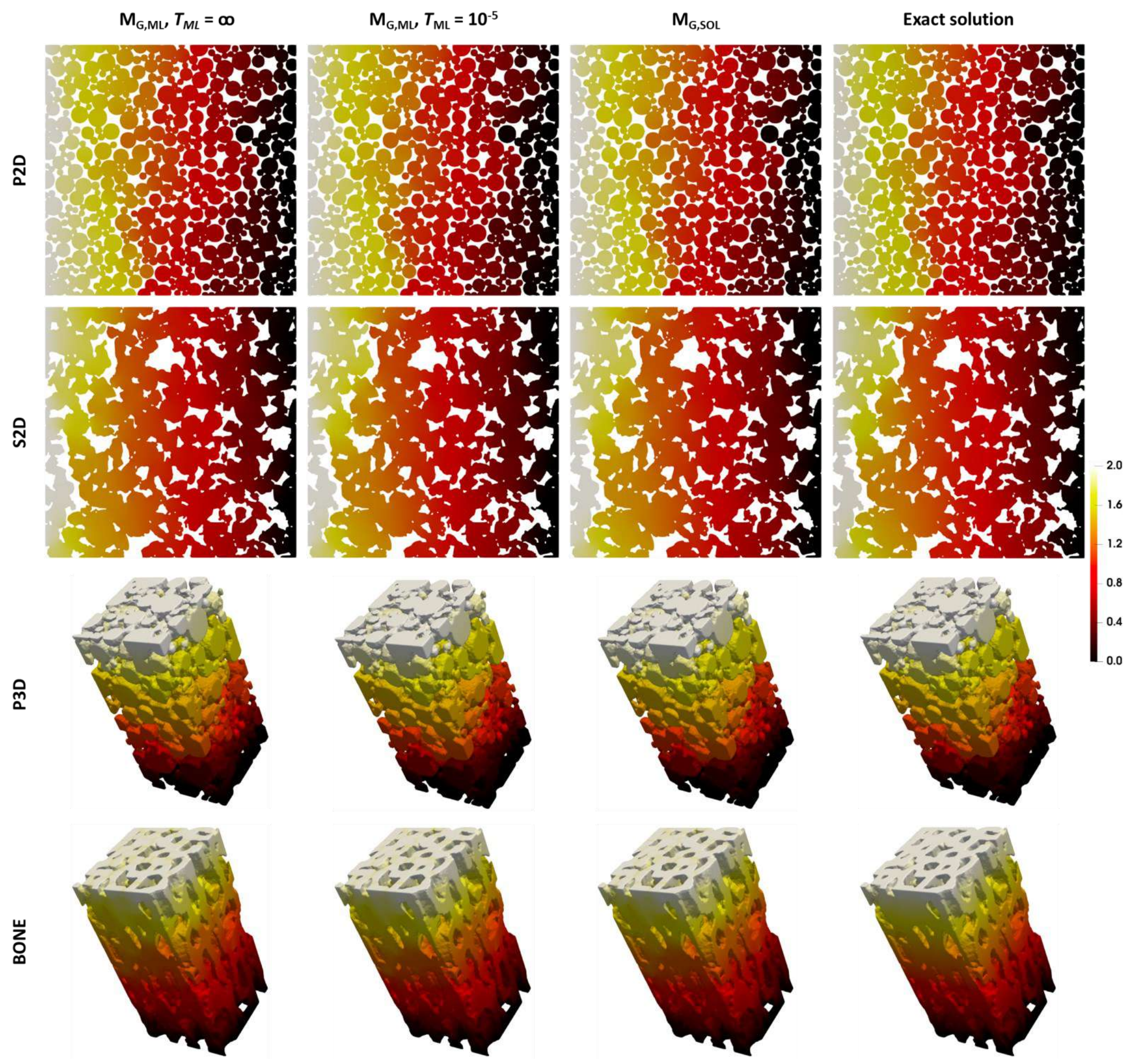

Figure 13: Comparison of the spatial distributions of u in the Poisson equation obtained from a single application of the coarse preconditioner, $\mathrm{M_G}$, against the exact solution over the P2D, S2D, P3D and BONE domains. The preconditioner $\mathrm{M_G}$ is computed via a numerical solver ($\mathrm{M_{G,SOL}}$), and the trained ML algorithm of Section 4 ($\mathrm{M_{G,ML}}$) with ($T_{ML} = 10^{-5}$) and without ($T_{ML} = \infty$) local smoothing of the basis vectors.

$\mathrm{M_{G,ML}}$. Since the costs of $\mathrm{M_{SOL}}$ and AMG do not depend on T_{ML}, they are depicted by the horizontal lines. In Fig.18 for the elasticity problem, AMG did not converge within 500 iterations (except for BONE). Therefore, the WCTs were extrapolated linearly based on the observed convergence rates, and distinguished by the dashed green lines.

We make the following observations: (1) In all cases, the cost of building $\mathrm{M_{G,ML}}$ with $T_{\mathrm{ML}} = \infty$ (almost identical to $T_{\mathrm{ML}} = 1$) is much lower than $\mathrm{M_{G,SOL}}$. For the Poisson problem, the speedup is a factor of 4.1 in P2D, 4.0 in S2D, 5.7 in P3D, and 1.6 in BONE, and for the elasticity problem, the speedup is a factor of 6.0 in P2D, 5.9 in S2D, 9.1 in P3D, and 3.1 in BONE; (2) As T_{ML} is reduced, the cost of building $\mathrm{M_{G,ML}}$ increases, while the solver cost decreases. In the Poisson problem, the decrease in solver cost is negligible and the absolute cost is comparable to $\mathrm{M_{SOL}}$. Hence, the total

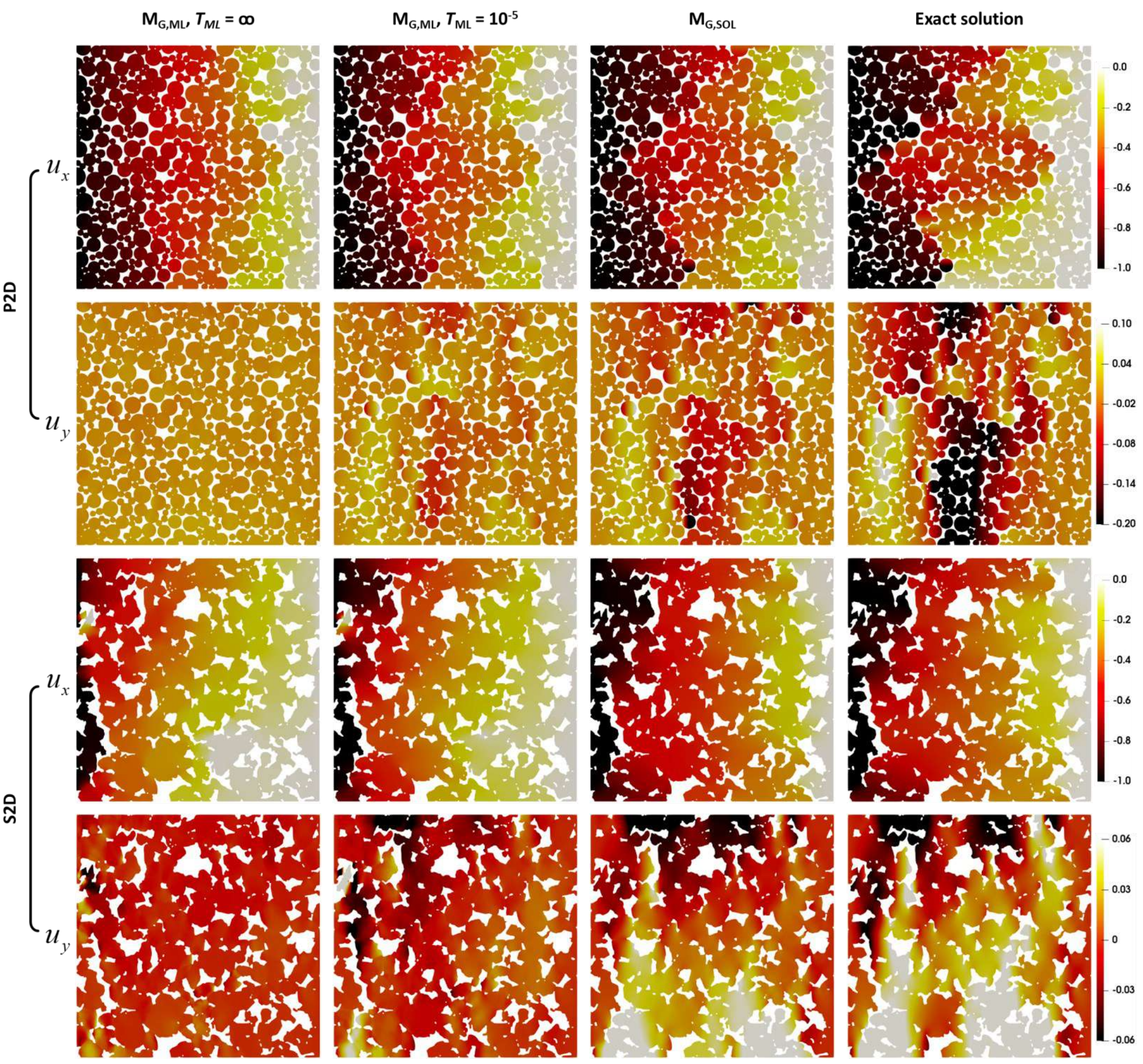

Figure 14: Comparison of the spatial distributions of u_x and u_y in the elasticity equation obtained from a single application of the coarse preconditioner, M_G, against the exact solution over the P2D and S2D domains. The preconditioner M_G is computed via a numerical solver ($M_{G,SOL}$), and the trained ML algorithm of Section 4 ($M_{G,ML}$) with ($T_{ML} = 10^{-5}$) and without ($T_{ML} = \infty$) local smoothing of the basis vectors.

cost (right column in Fig.17) increases monotonically as $T_{ML} \to 0$, implying $M_{G,ML}$ with $T_{ML} = \infty$ is recommended for preconditioning GMRES. In the elasticity problem, the decrease in solver cost as $T_{ML} \to 0$ is more noticeable, which causes the total cost to remain roughly flat as T_{ML} is varied (except for BONE, where the profile is U-shaped; although the y-axis range is narrow). Since T_{ML} plays a minor role in the total cost, we still recommend $M_{G,ML}$ with $T_{ML} = \infty$ for the elasticity problem; (3) Comparing the total costs of M_{SOL} and M_{ML} with $T_{ML} = \infty$, we see only moderate speedup (less than $\times 1.5$) with M_{ML} over M_{SOL} in the Poisson problem, and almost no speedup in the elasticity problem. This is because once the build-time of M_G is reduced by ML, the total cost is dominated by the solve-time of GMRES; especially in the elasticity problem. Another way to frame this is: *error control does not come*

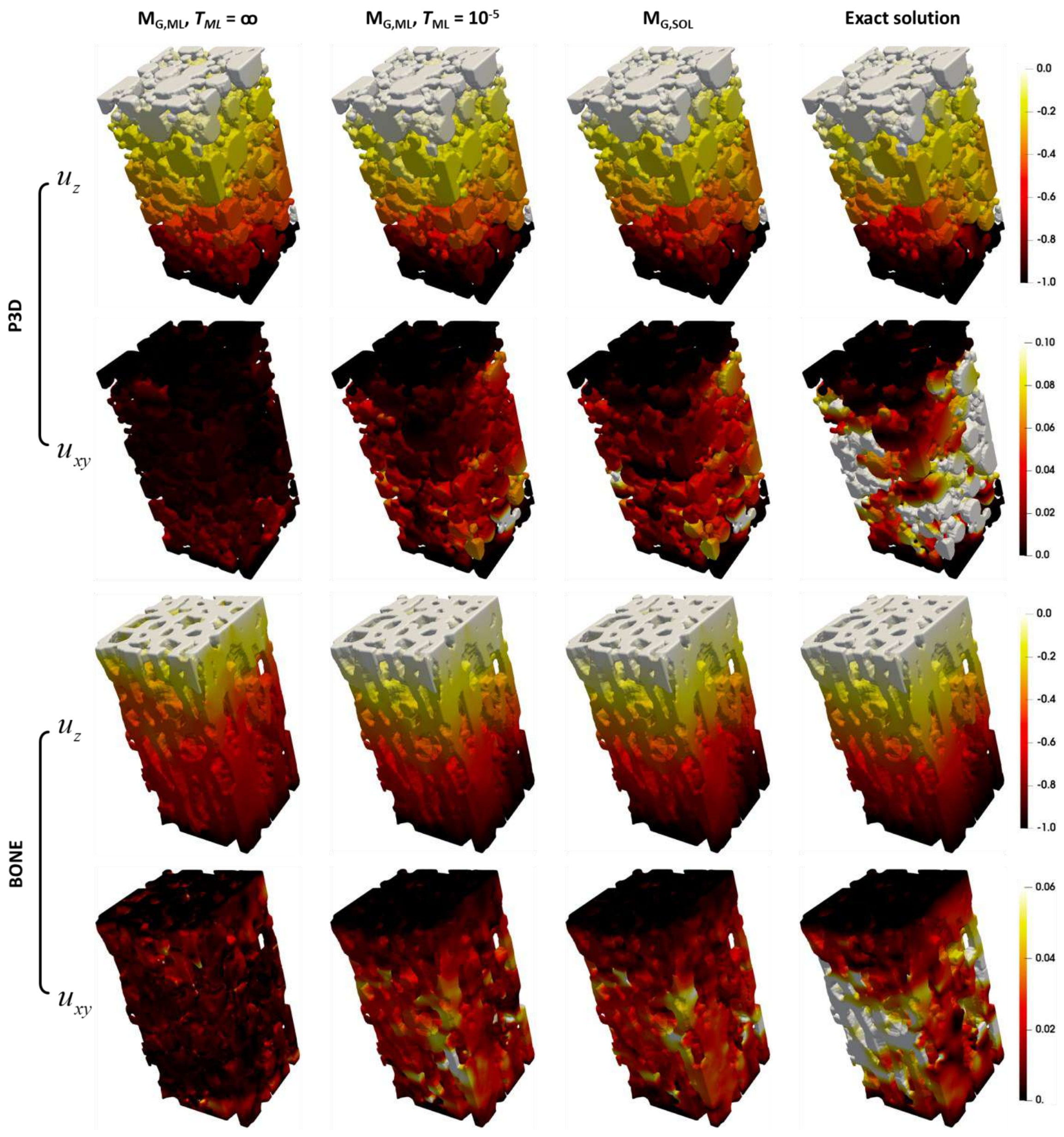

Figure 15: Comparison of the spatial distributions of u_z and $u_{xy} = \sqrt{u_x^2 + u_y^2}$ in the elasticity equation obtained from a single application of the coarse preconditioner, $\mathrm{M_G}$, against the exact solution on the P3D and BONE domains. The preconditioner $\mathrm{M_G}$ is computed via a numerical solver ($\mathrm{M_{G,SOL}}$), and the trained ML algorithm of Section 4 ($\mathrm{M_{G,ML}}$) with ($T_{ML} = 10^{-5}$) and without ($T_{ML} = \infty$) local smoothing of the basis vectors.

for free. If only a first-pass solution is desired (no GMRES iterations), ML can save time, but not if errors are to be reduced. Incidentally, a first-pass solution via $\mathrm{M_{G,ML}}$ with $T_{\mathrm{ML}} = \infty$ is comparable in CPU-time (but lower in memory footprint) to predictions from existing ML algorithms trained on whole domains (discussed in Section 7.1). We note the build-times of $\mathrm{M_{G,ML}}$ in Figs.17-18 include pre-processing costs associated with preparing the ML-input features (e.g., up/downscaling, smoothed tags via Eq.18) often excluded from prediction costs reported in the literature.

Finally, (4) in all cases except the Poisson problems defined on the P3D and BONE domains, the total cost of AMG is higher than both $\mathrm{M_{SOL}}$ and $\mathrm{M_{ML}}$ with $T_{\mathrm{ML}} = \infty$. The exceptions are largely due to AMG's fast build-time. For the elasticity problem, however, AMG's total cost is one to two orders of magnitude higher than either $\mathrm{M_{SOL}}$ and

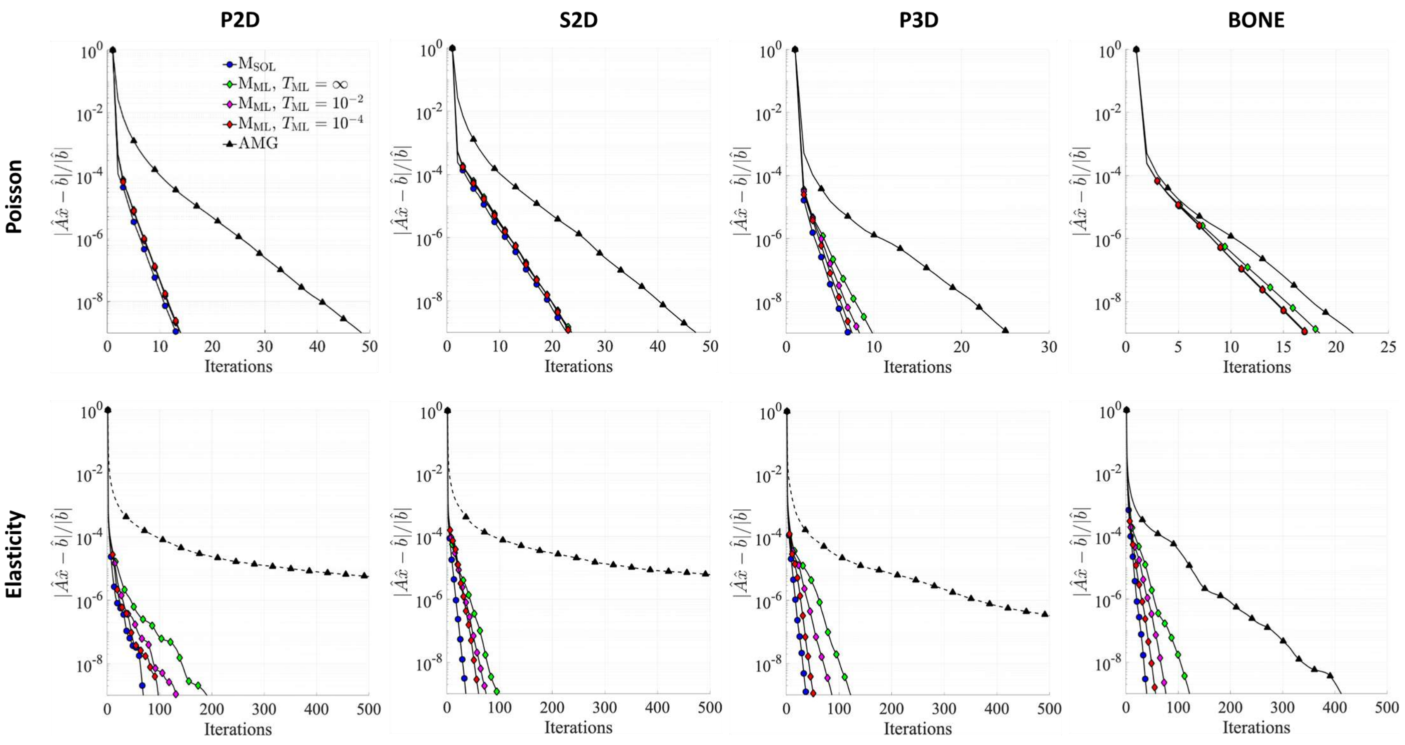

Figure 16: Normalized residual versus number of GMRES iterations preconditioned by AMG, M_{SOL}, and M_{ML} with $T_{ML} = \infty$, 10^{-2}, and 10^{-4} for the Poisson and elasticity problems on the P2D, S2D, P3D and BONE domains. $T_{ML} = \infty$ means no basis-smoothing iterations are performed.

M_{ML}. In much larger 3D domains than those of Fig.4, where subdomains consist of many more grids, we expect a clearer advantage of M_{ML} with $T_{ML} = \infty$ over M_{SOL} or AMG within GMRES. Moreover, the use of a cheaper smoother in series, like ILU(0), or parallelizing M_{CG} would reduce cost further. These claims remains to be substantiated.

7. Discussion

7.1. Approximate solutions with machine learning

In Section 3.3, we outlined a coarse preconditioner, M_G, based on the pore-level multiscale method (PLMM) that was proposed by [25, 26] for solving elliptic PDEs like the Poisson and elasticity Eqs.1 and 2. One can use M_G to obtain very accurate approximate, or *first-pass*, solutions to the discretized system $\hat{A}\hat{x} = \hat{b}$ via $x_{aprx} = M_G^{-1}\hat{b}$. In Section 4, we proceeded to show how the construction of M_G, requiring the computation of multiple basis functions on each subdomain, can be significantly accelerated by training ML algorithms. The latter were trained on a set of precomputed basis vectors defined on small, cropped images from larger, whole domains. The results in Section 6.2.1 demonstrated that first-pass solutions obtained from such a ML-built coarse preconditioner, $M_{G,ML}$, have comparable accuracy to those obtained from a solver-built one, $M_{G,SOL}$. Despite slightly larger errors in the tangential component of displacement in the elasticity problem with respect to the axial loading direction, the overall accuracy (<1% errors for 2D and <5% for 3D) and performance (×2-6 speedup for Poisson and ×3-9 for elasticity) of $M_{G,ML}$ were impressive. Basis smoothing was deemed unnecessary to obtain such predictions (i.e., $T_{ML} = \infty$ is sufficient).

Notice the approach of using ML to build basis functions on smaller subdomains, packaged in the form of a prolongation matrix P (Eq.14), is more advantageous than training ML algorithms on whole domains. As reasoned in Section 1, data curation is cheaper as 10^{2-3} subdomains can be obtained from decomposing one whole domain, training is faster because the overhead on computer memory is low, and the ML algorithm is simpler and less data hungry because the statistical space of possible subdomain geometries is much smaller than that of whole domains (e.g., subdomains are convex and devoid of holes or cavities). As stated in Section 6.2.3, the CPU-time (not memory footprint)

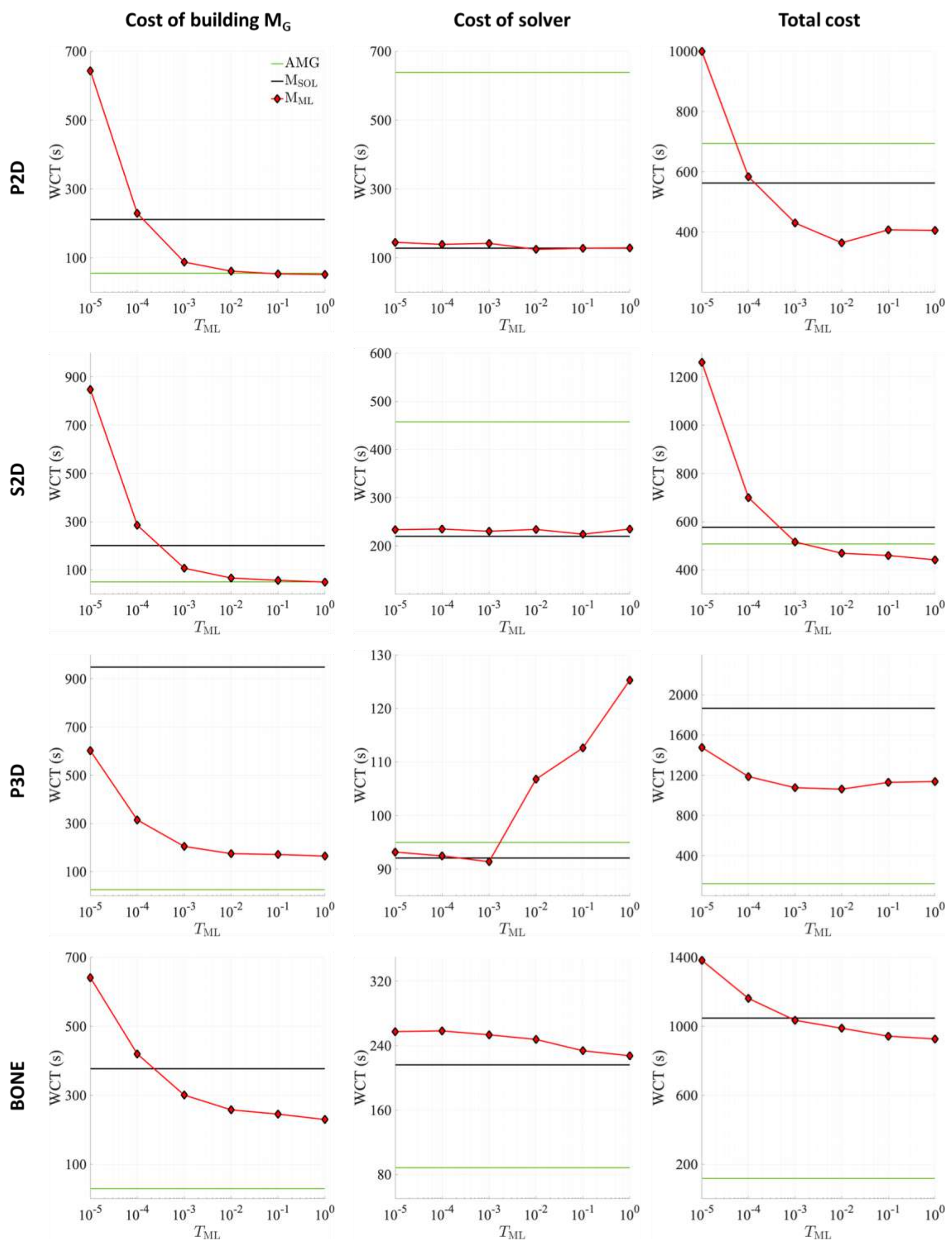

Figure 17: Wall-clock times (WCTs) in seconds associated with building the coarse preconditioner M_G (left column) and solving the linear system via GMRES to satisfy $\|\hat{A}\hat{x} - \hat{b}\| / \|\hat{b}\| < 10^{-9}$ (middle column) for the Poisson equation defined on all the domains. The total cost, including that of building the smoother in M_{SOL} and M_{ML}, is shown in the right column. All WCTs are plotted versus the tolerance (T_{ML}) used for basis smoothing iterations in the building of $M_{G,ML}$. Notice the costs of M_{SOL} and AMG do not depend on T_{ML} and are thus depicted by horizontal lines.

associated with *predicting* (not training) first-pass solutions via $M_{G,ML}$ with $T_{ML} = \infty$ is comparable to those of existing ML algorithms trained on whole domains. The seemingly higher than expected WCTs for building $M_{G,ML}$ ($T_{ML} = 1$) in Figs.17-18 are because they include pre-processing costs of the ML-input features (e.g., up/downsampling, smoothed

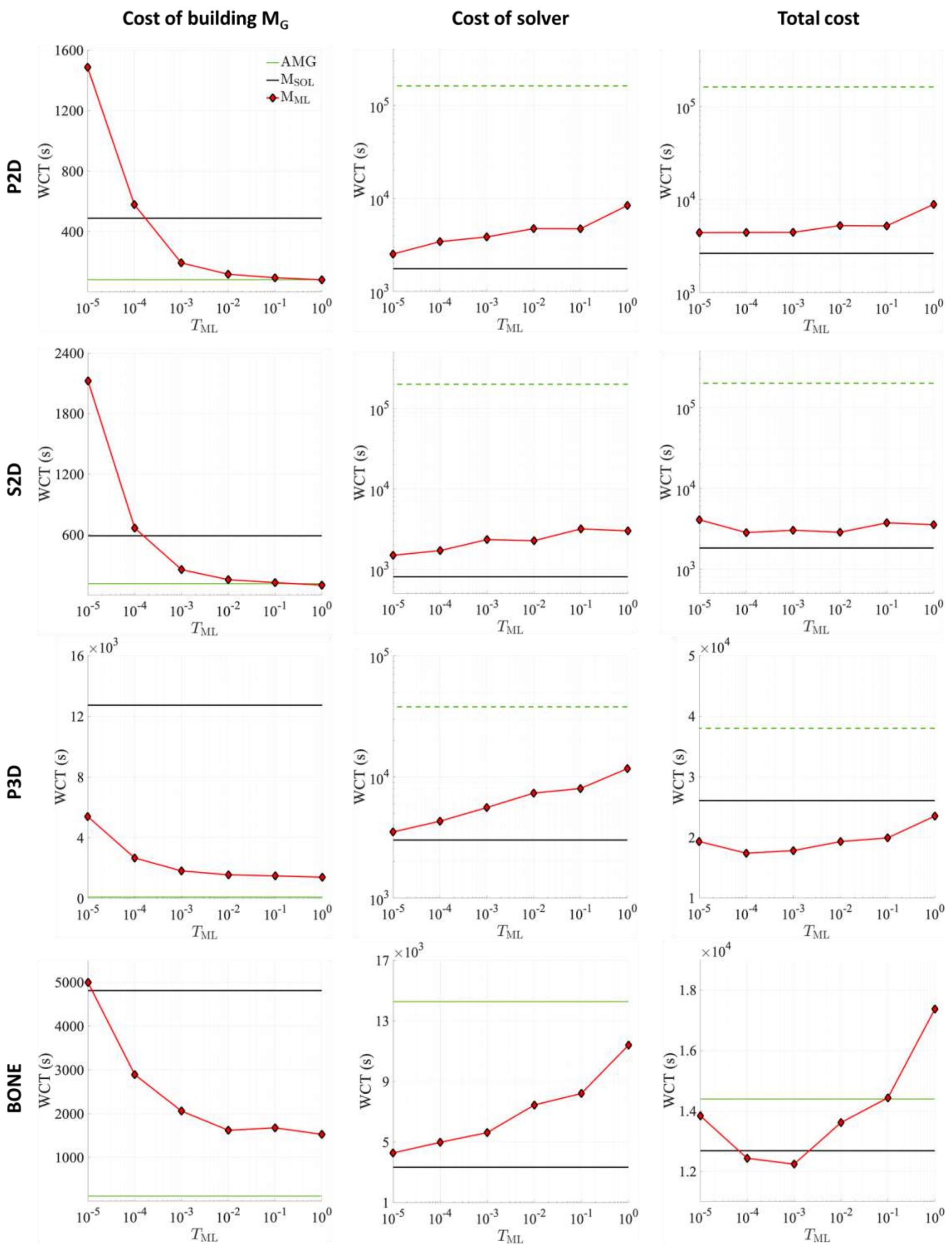

Figure 18: Wall-clock times (WCTs) in seconds associated with building the coarse preconditioner M_G (left column) and solving the linear system via GMRES to satisfy $\|\hat{A}\hat{x} - \hat{b}\| / \|\hat{b}\| < 10^{-9}$ (middle column) for the elasticity equation defined on all the domains. The total cost, including that of building the smoother, is shown in the right column. All WCTs are plotted against the tolerance (T_{ML}) used for basis smoothing iterations in the building of $M_{G,ML}$. Notice the costs of M_{SOL} and AMG do not depend on T_{ML} and are thus depicted by horizontal lines.

tag via Eq.18), often excluded from prediction costs in the literature. A final benefit of the approach herein is that by using a finite-volume restriction matrix, instead of the Galerkin $\hat{R} = \hat{P}^{\top}$ in Eq.9, one could impose global flux/stress conservation across all grain grids and contact interfaces. Future work will probe such a restriction matrix by [26].

7.2. Controlling the errors of machine learning

In addition to reducing the build-time of M_G via ML, we also succeeded in controlling the ML algorithms' errors. The latter, however, had interesting implications. Our approach consisted of two measures. First, local smoothing iterations were performed on each ML-predicted basis to reduce its high-frequency errors, up to tolerance T_{ML}, prior to building $M_{G,ML}$. We concluded this to be superfluous and recommended $M_{G,ML}$ be built with *no* basis smoothing (i.e., $T_{ML} = \infty$). This is because the accuracy and performance of $M_{G,ML}$ is already very good, and reducing T_{ML} only increases cost. Moreover, even at the limit $T_{ML} \rightarrow 0$, $M_{G,ML}$ is only as good as $M_{G,SOL}$, which itself incurs high-frequency errors. Our second, and more important, measure was to pair $M_{G,ML}$ with a smoother M_L, for which we picked the compatible M_{CG} recommended by [26] and introduced in Section 3.4. Still adhering to $T_{ML} = \infty$, the convergence rate of GMRES preconditioned by M_{ML} (combining $M_{G,ML}$ with M_{CG}) is comparable to M_{SOL} (combining $M_{G,ML}$ with M_{CG}) in the Poisson problem, but lower by up to $\times 2$ in the Elasticity problem. Reducing T_{ML} reduces the cost of GMRES for the elasticity problem, but increases the build-time of $M_{G,ML}$. Overall, the total cost of M_{ML}, consisting of building $M_{G,ML}$, setting up M_{CG}, and solving GMRES is comparable or only slightly lower than M_{SOL}.

This could have several contributors: (1) Once the build-time of M_G is reduced, the remaining WCT is dominated by GMRES, especially for the elasticity problem. This is likely because we demand such a low tolerance for error (= 10^{-9}). A larger tolerance would result almost certainly in better speedups for M_{ML} over M_{SOL}, as is the case for the first-pass solutions via $M_{G,ML}$ with $T_{ML} = \infty$; (2) The setup cost of M_{CG}, included in the total WCTs of M_{ML} and M_{SOL}, is rather high, as it consists of LU decompositions performed on local systems defined on the grain and contact grids. Building M_{CG} in parallel, or using a cheaper smoother in series (e.g., ILU(0)) would have reduced the total WCT, but by an equal amount for both M_{ML} and M_{SOL}; (3) The ML algorithms of Section 4 could have benefited from longer training, given the errors are still decreasing after 200 epochs in Fig.B.1, albeit slowly. This could have resulted in lower GMRES cost, thus smaller total WCT, especially in the elasticity problem; and (4) ML algorithms, particularly of the "deep convolutional" kind, are not as cheap as celebrated. Each layer entails a matrix-vector multiplication, and the deeper the network, the more multiplications are required. This is not unlike iterations in a linear solver, and if the goal is to beat a rapidly converging preconditioner like M_{SOL}, there is a limit to a network's depth. Despite said issues, M_{ML} (and M_{SOL}) outperforms AMG by $\times 10^{1-2}$ in the elasticity, but not the more well-trodden Poisson, problem.

We highlight that controlling and estimating prediction errors is not free of cost. ML algorithms are often used to perform blind predictions, whose error bounds are guaranteed empirically and statistically at best (i.e., based on a testing set). While appropriate for applications like uncertainty quantification, for deterministic predictions, error bounds lack. What this work suggests is that some combination of ML with a physics-solver is needed to fill this gap, which involves corrective iterations. The iterations increase cost, but also confidence in the results. If the merit of an algorithm were placed solely on the swiftness of its output, then the first-pass solution via $M_{G,ML}$ and $T_{ML} = \infty$ accomplishes what most methods can with fewer data and shorter training time (see Section 7.1). But if added merit were placed on the reliability of said outputs, as we think one should, then our approach offers a path forward.

7.3. Transferability across geometries and material properties

A pleasantly surprising observation, also made by others (e.g., [40]), is that the ML algorithm trained in Section 4 on disk/sphere packs also applies to other subdomain geometries. Moreover, in the elasticity problem, training was performed by assuming a Poisson's ratio of ν=0.08 and yet, the ML algorithm predicts with reasonable accuracy bases for ν=0.4 and 0.45. Notice one need not probe a second elastic modulus, like λ or μ, in addition to ν. This is because once ν is fixed, the other moduli can be matched by simply scaling the ML-predicted bases. In Section 6.1, basis errors nearly doubled when the ML algorithms were tested on out-of-distribution geometries or ν. But those errors were still <20% and resulted in high-quality first-pass solutions, as listed in Table 2, and rapid GMRES convergence, as shown in Figs.16 and C.4. This hints at the promising outlook that pre-training a library of ML algorithms for different grain shapes, material properties, and even PDEs can be readily applied to, or minimally transfer-trained on new problems. The latter may even be done on-the-fly, while solving a time-dependent or many-query problem relevant to optimization or uncertainty quantification with respect to variability in the porous microstructure.

7.4. Other machine learning methods and architectures

In Appendix B, we included the L_2-norm of the PDE residual in the loss function as specified by Eq.19. This is similar to physics-informed neural networks (PINNs) [38, 55, 56], wherein a PDE's residual and BCs are "softly imposed" through the loss function; as opposed to a "hard imposition" via non-trainable layers. The weights α_a and α_e in Eq.19 control the emphasis on the PDE mismatch over the data mismatch. Figs.B.1-B.2 show that the impact of the PDE mismatch is negligible on both the training speed and predictive accuracy of the ML algorithms. Ultimately, an ideal algorithm is one that incurs low errors, including those with high-frequency mode, eliminating the need for basis smoothing iterations altogether (Section 4.4). While improving the architecture in Fig.3, by perhaps appending non-trainable layers to it, is one option [57], adopting whole new emerging architectures such as graph neural networks (GNNs) [58, 59] or neural operators [60, 61] is another. Irrespective of the approach, however, close attention must be paid to the spectral properties of the errors in the basis functions produced by such ML algorithms.

8. Conclusion

We have presented an approach for using machine learning (ML) to build two-level preconditioners for efficiently solving elliptic equations on complex porous geometries via iterative solvers. While the preconditioner is based on the pore-level multiscale method (PLMM), the proposed approach applies to others based on domain decomposition or multigrid techniques. The PLMM preconditioner consists of a coarse preconditioner $\mathrm{M_G}$ and a smoother $\mathrm{M_L}$. We use ML to accelerate the construction of $\mathrm{M_G}$ only, as the U-Net architecture employed herein incurs high-frequency errors itself that are in need of smoothing. The overall framework can be viewed as: (1) *a way to remove a major bottleneck in building multiscale preconditioners*; or (2) *a robust mechanism to equip ML with error control capabilities*.

The proposed ML algorithms take cropped subdomain images as inputs and yield basis functions that satisfy a PDE locally as outputs, which are then used to assemble a prolongation matrix for $\mathrm{M_G}$. The training of ML algorithms on small subdomains is more advantageous than on whole domains because data generation is cheaper, the statistical space of data is smaller, training is faster and requires less memory, and basis functions can be assembled in arbitrary ways to satisfy any BCs or reused to solve similar (e.g., linearized, perturbed) PDEs. We tested the ML-built $\mathrm{M_{G,ML}}$ in solving the Poisson and linear-elasticity equations on challenging 2D/3D geometries and compared its performance against the solver-built $\mathrm{M_{G,SOL}}$ and AMG. We showed that $\mathrm{M_{G,ML}}$ can be used in standalone fashion to obtain approximate, or first-pass, solutions (without iterations) at a very low cost and almost the same accuracy as $\mathrm{M_{G,SOL}}$, which is useful in many practical applications (e.g., subsurface engineering). Moreover, when applied within GMRES, $\mathrm{M_{G,ML}}$ performed on par or slightly worse than $\mathrm{M_{G,SOL}}$, indicating its high quality as a preconditioner. However, the *total cost* saved with $\mathrm{M_{G,ML}}$ was not significant, because once the build-time of $\mathrm{M_G}$ had been reduced, the solver-time dominated given our very low tolerance imposed on GMRES (i.e., 10^{-9}). Increasing this tolerance, or applying ML to much larger 3D domains, is expected to render cost savings more apparent. Finally, both $\mathrm{M_{G,ML}}$ and $\mathrm{M_{G,SOL}}$ performed far better than AMG in the elasticity problem, but worse in the more well-trodden Poisson problem.

Future work on other ML architectures like neural operators or graph neural networks, with special attention to error spectra, are promising areas of research. While all computations here were in *series*, $\mathrm{M_{G,ML}}$ is fully parallelizable.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CMMI-2145222. We acknowledge the Institute for Computational and Data Sciences (ICDS) at Penn State University for access to computational resources. We also thank Amelie Sas at KU Leuven, Belgium, for providing the bone image in Fig.4.

Appendix A. Coarsening details of the AMG preconditioner

Table A.1 summarizes the number of coarsening levels and the coarsest matrix achieved by the AMG preconditioner applied to the Poisson and elasticity problems defined on the P2D, S2D, P3D, and BONE domains.

Appendix B. Impact of physics-informed loss functions on basis vectors

In Section 6.1, we probed the basis vectors constructed by ML algorithms trained with $\alpha_p = \alpha_e = 0$ in Eq.19. In other words, the loss functions were uninformed by the PDEs' residuals and training was purely data-driven. Here,

Table A.1: Summary of the number of coarsening levels and the size of the coarsest matrix achieved in the AMG preconditioner when applied to the Poisson and elasticity problems defined on the P2D, S2D, P3D and BONE domains.

		P2D	S2D	P3D	BONE
Poisson	No. levels	8	8	8	8
	Coarsest matrix	280×280	242×242	30×30	22×22
Elasticity	No. levels	8	8	8	8
	Coarsest matrix	428×428	361×361	50×50	37×37

we set $\alpha_p = 1$ and 10 while training the ML algorithm of the Poisson equation. The higher α_p is, the more physics-informed the loss function becomes. Our goal is to assess whether the training speed and/or the algorithm's testing accuracy are improved by increasing α_p. Fig.B.1 illustrates the training loss versus the number of training epochs for $\alpha_p = 0$, 1, and 10. When $\alpha_p = 0$, the training loss stagnates up to epoch 50, then drops rapidly afterwards. Increasing α_p to 1 or 10 has minimal impact on this stagnation period, reducing it only slightly to epoch 35.

As for the L_2-errors, Fig.B.2 shows errors corresponding to the predicted basis vectors of the P2D domain. We see $\alpha_p = 0$ and 1 yield basis vectors with similar accuracy, while $\alpha_p = 10$ is slightly worse. More precisely, the percentage of subdomains with errors below 0.05 are 86.1%, 86.5% and 81.9% for $\alpha_p = 0$, 1, and 10, respectively. While clearer benefits may be observed from including PDE residuals within loss functions in training larger ML architectures over many more epochs, for the cases studied herein, we do not observe such benefits.

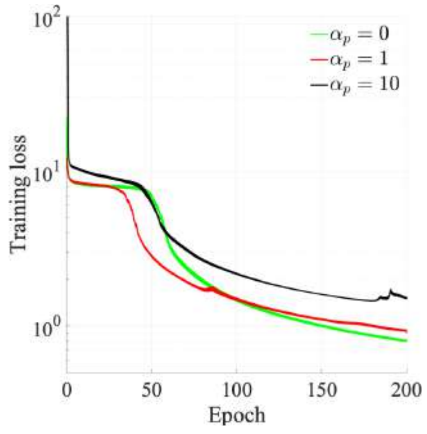

Figure B.1: Training loss versus number of training epochs for $\alpha_p = 0$, 1, and 10 in the loss function of the Poisson equation (Eq.19). The higher α_p is, the more physics-informed the loss function becomes. The $\alpha_p = 0$ case corresponds to purely data-driven training. A moving average, with a window size of 100 points, was used to dampen the significant oscillations observed in the training losses.

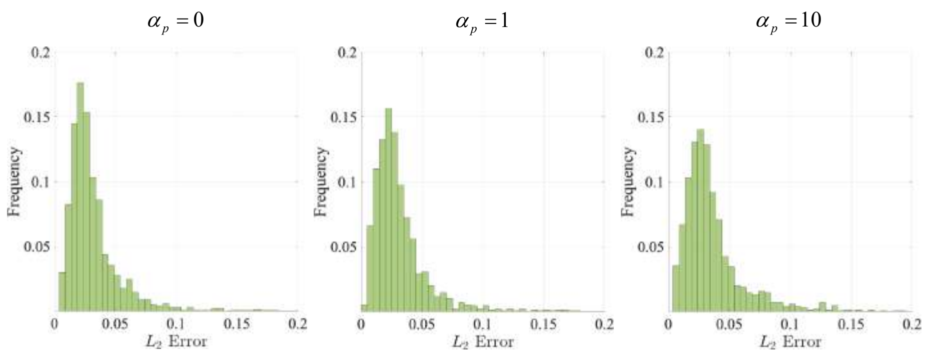

Figure B.2: PDFs of L_2-errors associated with the ML-predicted basis vectors for the Poisson problem defined on subdomains of the P2D domain (they constitute unseen data). Three ML algorithms are trained with $\alpha_p = 0$, 1, and 10 in their loss functions defined by Eq.19.

Appendix C. Transferability to different Poisson ratios of the elasticity problem

In Section 4, we trained ML algorithms for the elasticity equation assuming a single Poisson's ratio of $\nu = 0.08$, corresponding to α-quartz. Here, we determine whether these algorithms apply to other ν without having to retrain them. Specifically, we consider $\nu = 0.4$ and 0.45 and predict all basis vectors associated with the subdomains of the P2D and S2D domains. Fig.C.3 shows the PDFs of the L_2-errors for these bases, computed via Eq.23, where we see they are roughly twice as large for $\nu = 0.4$ and 0.45 than $\nu = 0.08$, but still mostly <20%. Fig.C.4 further shows the convergence rate of GMRES preconditioned by M_{ML}, whose coarse preconditioner $\mathrm{M}_{\mathrm{G,ML}}$ is built by the predicted basis vectors above. In other words, $\mathrm{M}_{\mathrm{G,ML}}$ is built using the ML algorithms of Section 4, which assume $\nu = 0.08$, and then applied to solve the elasticity problem defined on P2D and S2D with Poisson's ratios $\nu = 0.08$, 0.4, and 0.45. We see the convergence rate is minimally affected by varying ν, implying the ML algorithms of Section 4 are transferable to other mechanical properties. In Fig.C.4, no local smoothing of basis vectors was employed (i.e., $T_{ML} = \infty$).

References

[1] Stefan Bachu. Co2 storage in geological media: Role, means, status and barriers to deployment. Progress in energy and combustion science, 34(2):254–273, 2008.

[2] Angela Goodman Hanson, Barbara Kutchko, Greg Lackey, Djuna Gulliver, Brian R Strazisar, Kara A Tinker, Foad Haeri, Ruishu Wright, Nicolas Huerta, Seunghwan Baek, et al. Subsurface hydrogen and natural gas storage: State of knowledge and research recommendations report. 2022.

[3] Enrico Barbier. Geothermal energy technology and current status: an overview. Renewable and sustainable energy reviews, 6(1-2):3–65, 2002.

[4] Martin Andersson, SB Beale, M Espinoza, Z Wu, and W Lehnert. A review of cell-scale multiphase flow modeling, including water management, in polymer electrolyte fuel cells. Applied Energy, 180:757–778, 2016.

[5] Jason K Lee, ChungHyuk Lee, Kieran F Fahy, Pascal J Kim, Kevin Krause, Jacob M LaManna, Elias Baltic, David L Jacobson, Daniel S Hussey, and Aimy Bazylak. Accelerating bubble detachment in porous transport layers with patterned through-pores. ACS Applied Energy Materials, 3(10):9676–9684, 2020.

[6] Dorthe Wildenschild and Adrian P Sheppard. X-ray imaging and analysis techniques for quantifying pore-scale structure and processes in subsurface porous medium systems. Advances in Water Resources, 51:217–246, 2013.

[7] Veerle Cnudde and Matthieu Nicolaas Boone. High-resolution x-ray computed tomography in geosciences: A review of the current technology and applications. Earth-Science Reviews, 123:1–17, 2013.

[8] Paul Meakin and Alexandre M Tartakovsky. Modeling and simulation of pore-scale multiphase fluid flow and reactive transport in fractured and porous media. Reviews of Geophysics, 47(3), 2009.

[9] Yousef Saad. Iterative methods for sparse linear systems, volume 82. siam, 2003.

[10] John W Ruge and Klaus Stüben. Algebraic multigrid. In Multigrid methods, pages 73–130. SIAM, 1987.

[11] Yvan Notay. An aggregation-based algebraic multigrid method. Electronic transactions on numerical analysis, 37(6):123–146, 2010.

[12] Hui Zhou and Hamdi A Tchelepi. Operator-based multiscale method for compressible flow. SPE Journal, 13(02):267–273, 2008.

[13] TH Sandve, E Keilegavlen, and JM Nordbotten. Physics-based preconditioners for flow in fractured porous media. Water Resources Research, 50(2):1357–1373, 2014.

[14] Nicola Castelletto, Hadi Hajibeygi, and Hamdi A Tchelepi. Multiscale finite-element method for linear elastic geomechanics. Journal of Computational Physics, 331:337–356, 2017.

[15] Fanxiang Xu, Hadi Hajibeygi, and Lambertus J Sluys. Multiscale extended finite element method for deformable fractured porous media. Journal of Computational Physics, 436:110287, 2021.

[16] Hui Zhou and Hamdi A Tchelepi. Two-stage algebraic multiscale linear solver for highly heterogeneous reservoir models. SPE Journal, 17 (02):523–539, 2012.

[17] Yixuan Wang, Hadi Hajibeygi, and Hamdi A Tchelepi. Algebraic multiscale solver for flow in heterogeneous porous media. Journal of Computational Physics, 259:284–303, 2014.

[18] Todd Arbogast and Hailong Xiao. Two-level mortar domain decomposition preconditioners for heterogeneous elliptic problems. Computer Methods in Applied Mechanics and Engineering, 292:221–242, 2015.

[19] Ivo Babuška and John E Osborn. Generalized finite element methods: their performance and their relation to mixed methods. SIAM Journal on Numerical Analysis, 20(3):510–536, 1983.

[20] Thomas Y Hou and Xiao-Hui Wu. A multiscale finite element method for elliptic problems in composite materials and porous media. Journal of computational physics, 134(1):169–189, 1997.

[21] Patrick Jenny, SH Lee, and Hamdi A Tchelepi. Multi-scale finite-volume method for elliptic problems in subsurface flow simulation. Journal of Computational Physics, 187(1):47–67, 2003.

[22] Hadi Hajibeygi, Giuseppe Bonfigli, Marc Andre Hesse, and Patrick Jenny. Iterative multiscale finite-volume method. Journal of Computational Physics, 227(19):8604–8621, 2008.

[23] Christine Bernardi, Y Maday, and A T Patera. A new nonconforming approach to domain decomposition: the mortar element method. Nonlinear partial equations and their applications, 1994.

[24] Todd Arbogast, Lawrence C Cowsar, Mary F Wheeler, and Ivan Yotov. Mixed finite element methods on nonmatching multiblock grids. SIAM Journal on Numerical Analysis, 37(4):1295–1315, 2000.

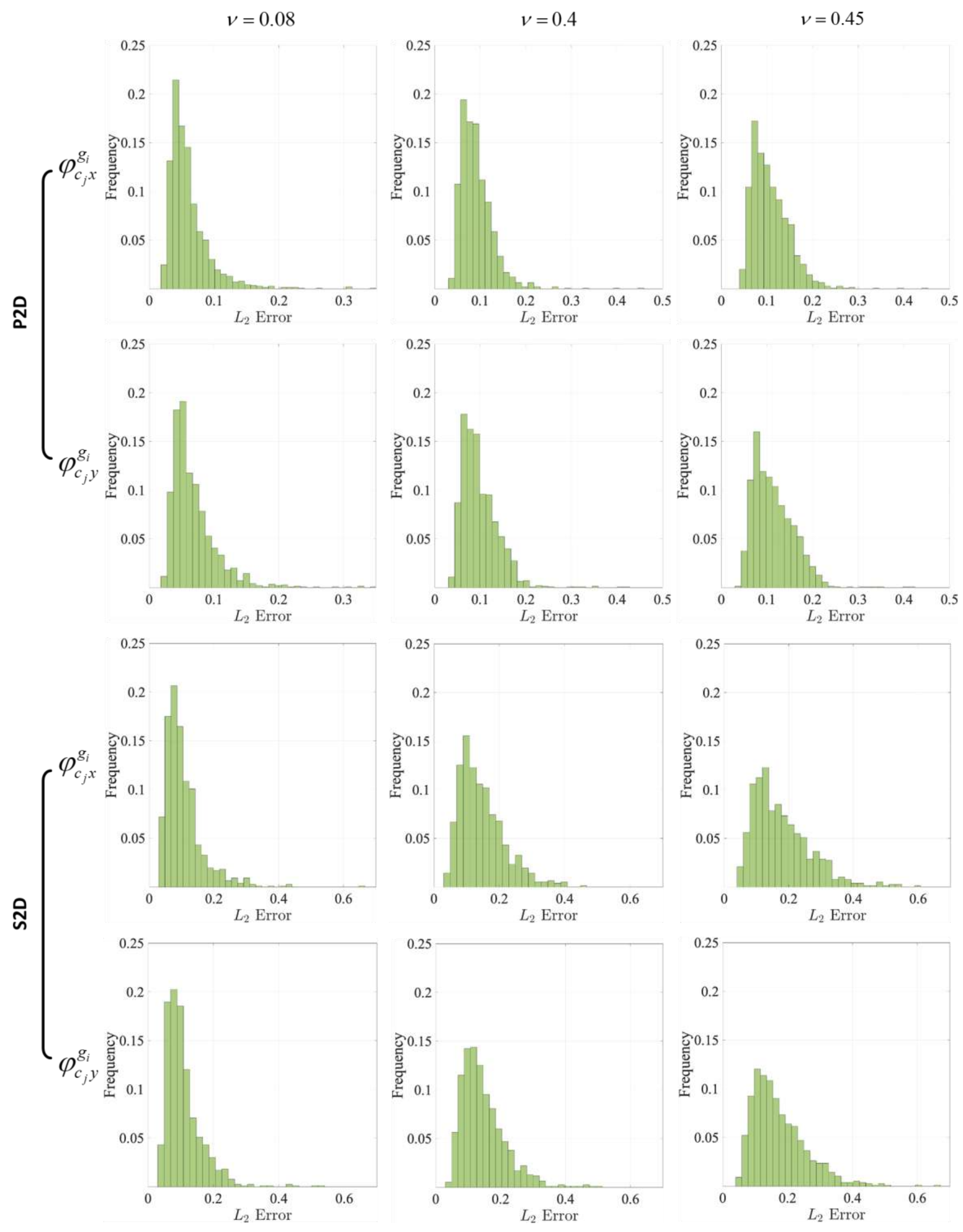

Figure C.3: PDFs of L_2-errors associated with the ML-predicted basis vectors for the elasticity problem defined on subdomains of the P2D and S2D domains with Poisson's ratios ν = 0.08, 0.4, and 0.45 (they constitute unseen data). The ML algorithms are the same as those trained in Section 4 on data that assumed ν = 0.08. In other words, the training set is uninformed by ν = 0.4 and 0.45.

[25] Yashar Mehmani and Kangan Li. A multiscale preconditioner for microscale deformation of fractured porous media. Journal of Computational Physics, 482:112061, 2023.

[26] Kangan Li and Yashar Mehmani. A multiscale preconditioner for crack evolution in porous microstructures: Accelerating phase-field methods. International Journal for Numerical Methods in Engineering, In Press, 2024.

[27] Yashar Mehmani, Nicola Castelletto, and Hamdi A Tchelepi. Multiscale formulation of frictional contact mechanics at the pore scale. Journal of Computational Physics, 430:110092, 2021.

[28] Kangan Li and Yashar Mehmani. A pore-level multiscale method for the elastic deformation of fractured porous media. Journal of Computational Physics, 483:112074, 2023.

[29] Yashar Mehmani and Kangan Li. Multiscale preconditioning of stokes flow in complex porous geometries. Journal of Computational Physics, Under Review, 2024.

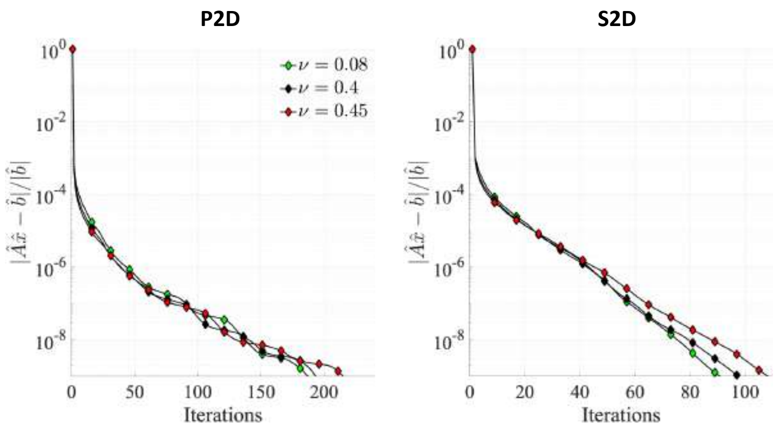

Figure C.4: Normalized residual versus GMRES iterations for the linear system, Eq.6, of the elasticity problem defined on the P2D and S2D domains with Poisson's ratios ν = 0.08, 0.4, or 0.45. GMRES is preconditioned by $\mathrm{M_{ML}}$, whose coarse preconditioner $\mathrm{M_G}$ is constructed using the ML algorithms of Section 4, which assume ν=0.08 for all subdomains in the training set. No basis smoothing is employed ($T_{ML}=\infty$).

[30] Yashar Mehmani and Hamdi A Tchelepi. Multiscale computation of pore-scale fluid dynamics: Single-phase flow. Journal of Computational Physics, 375:1469–1487, 2018.

[31] Yashar Mehmani and Hamdi A Tchelepi. Multiscale formulation of two-phase flow at the pore scale. Journal of Computational Physics, 389: 164–188, 2019.

[32] Bo Guo, Yashar Mehmani, and Hamdi A Tchelepi. Multiscale formulation of pore-scale compressible darcy-stokes flow. Journal of Computational Physics, 397:108849, 2019.

[33] Sabit Mahmood Khan, Kangan Li, and Yashar Mehmani. Order reduction of fracture mechanics in porous microstructures: A multiscale computing framework. Computer Methods in Applied Mechanics and Engineering, 420:116706, 2024.

[34] Serge Beucher and Christian Lantuéjoul. Use of watersheds in contour detection. In International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation, Rennes, France, 1979.

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, pages 234–241. Springer, 2015.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[37] Pejman Tahmasebi, Serveh Kamrava, Tao Bai, and Muhammad Sahimi. Machine learning in geo-and environmental sciences: From small to large scale. Advances in Water Resources, 142:103619, 2020.

[38] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. Nature Reviews Physics, 3(6):422–440, 2021.

[39] Javier E Santos, Duo Xu, Honggeun Jo, Christopher J Landry, Maša Prodanović, and Michael J Pyrcz. Poreflow-net: A 3d convolutional neural network to predict fluid flow through porous media. Advances in Water Resources, 138:103539, 2020.

[40] Javier E Santos, Ying Yin, Honggeun Jo, Wen Pan, Qinjun Kang, Hari S Viswanathan, Maša Prodanović, Michael J Pyrcz, and Nicholas Lubbers. Computationally efficient multiscale neural networks applied to fluid flow in complex 3d porous media. Transport in porous media, 140(1):241–272, 2021.

[41] Serveh Kamrava, Pejman Tahmasebi, and Muhammad Sahimi. Physics- and image-based prediction of fluid flow and transport in complex porous membranes and materials by deep learning. Journal of Membrane Science, 622:119050, 2021.

[42] Xu-Hui Zhou, James E McClure, Cheng Chen, and Heng Xiao. Neural network–based pore flow field prediction in porous media using super resolution. Physical Review Fluids, 7(7):074302, 2022.

[43] Ying Da Wang, Traiwit Chung, Ryan T Armstrong, and Peyman Mostaghimi. Ml-lbm: Predicting and accelerating steady state flow simulation in porous media with convolutional neural networks. Transport in Porous Media, 138(1):49–75, 2021.

[44] Eduardo A de Souza Neto, Djordje Peric, and David RJ Owen. Computational methods for plasticity: theory and applications. John Wiley & Sons, 2011.

[45] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics, 394:56–81, 2019.

[46] Govinda Anantha Padmanabha and Nicholas Zabaras. A bayesian multiscale deep learning framework for flows in random media. arXiv preprint arXiv:2103.09056, 2021.

[47] Olav Møyner and Knut-Andreas Lie. A multiscale restriction-smoothed basis method for high contrast porous media represented on unstructured grids. Journal of Computational Physics, 304:46–71, 2016.

[48] Jean Serra. Introduction to mathematical morphology. Computer vision, graphics, and image processing, 35(3):283–305, 1986.

[49] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. Road extraction by deep residual u-net. IEEE Geoscience and Remote Sensing Letters, 15(5):749–753, 2018.

[50]

[51] Juncai He and Jinchao Xu. Mgnet: A unified framework of multigrid and convolutional neural network. Science china mathematics, 62: 1331–1354, 2019.

[52] Steffen Berg, Ryan Armstrong, and Andreas Wiegmann. Gildehauser sandstone. `http://www.digitalrocksportal.org/projects/134`, 2018.

[53] Amelie Sas, Benedikt Helgason, Stephen J Ferguson, and G Harry van Lenthe. Mechanical and morphological characterization of pmma/bone composites in human femoral heads. Journal of the Mechanical Behavior of Biomedical Materials, 115:104247, 2021.

[54] Long Chen. *i*FEM: an integrated finite element methods package in MATLAB. Technical report, 2009. URL `https://github.com/lyc102/ifem`.

[55] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. arXiv preprint arXiv:1711.10561, 2017.

[56] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics, 378:686–707, 2019.

[57] Arvind T Mohan, Nicholas Lubbers, Daniel Livescu, and Michael Chertkov. Embedding hard physical constraints in neural network coarse-graining of 3d turbulence. arXiv preprint arXiv:2002.00021, 2020.

[58] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. IEEE transactions on neural networks, 20(1):61–80, 2008.

[59] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In International Conference on Machine Learning, pages 6489–6499. PMLR, 2020.

[60] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895, 2020.

[61] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. arXiv preprint arXiv:2108.08481, 2021.